NASoftware Limited

Incorporating InfoSAR

CSIPL
Reference Manual

CSIPL/Ref [3.20.14]

Release 3.20.14
February 2021

This document is derived from the VSIPL API 1.01 Specification document written by David A. Schwartz (HRL Laboratories, LLC), Randall R. Judd (Space and Naval Warfare Systems Center, San Diego), William J. Harrod (Silicon Graphics Inc./Cray Research) and Dwight P. Manley (Compaq Computer Corp./Digital Equipment Corp.), approved by the VSIPL Forum (27 March 2001), and available from `http://www.vsipl.org/`.

The copyright statement of the original document follows:

©1999–2000 Georgia Tech Research Corporation, all rights reserved.

A non-exclusive, non-royalty bearing license is hereby granted to all persons to copy, modify, distribute and produce derivative works for any purpose, provided that this copyright notice and following disclaimer appear on all copies: THIS LICENSE INCLUDES NO WARRANTIES, EXPRESSED OR IMPLIED, WHETHER ORAL OR WRITTEN, WITH RESPECT TO THE SOFTWARE OR OTHER MATERIAL INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF PERFORMANCE OR DEALING, OR FROM USAGE OR TRADE, OR OF NON-INFRINGEMENT OF ANY PATENTS OF THIRD PARTIES. THE INFORMATION IN THIS DOCUMENT SHOULD NOT BE CONSTRUED AS A COMMITMENT OF DEVELOPMENT BY ANY OF THE ABOVE PARTIES.

This material is based in part upon work supported by Ft. Huachuca/DARPA under contract No. DABT63-96-C-0060. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Ft. Huachuca/DARPA.

The US Government has a license under these copyrights, and this material may be reproduced by or for the US Government.

The derived document is Copyright N.A.Software 2015.

Contents

1 Introduction **1**

1.1 Types . . . 1
1.2 Symbols and Flags . . . 1
1.3 Vector Variables . . . 1
1.4 Matrix Variables . . . 2
1.5 Complex Variables . . . 2
1.6 Function Names . . . 2
1.7 Hints . . . 3
1.8 Notation . . . 3
1.9 Errors and Restrictions . . . 3

2 Getting the Best Performance **5**

2.1 Version Information . . . 5
2.2 Memory Alignment . . . 5
2.3 Vector/Matrix Format . . . 5
2.4 Error Checking and Debugging . . . 6
2.5 Support Functions . . . 7
2.6 Scalar Functions . . . 7
2.7 Random Number Generation . . . 7
2.8 Vector and Elementwise Operations . . . 7
2.9 Signal Processing Functions . . . 7
2.10 FFT Functions . . . 8
2.11 FIR Filter, Convolution and Correlation Functions . . . 8
2.12 Linear Algebra Functions . . . 8
2.13 Matrix and Vector Operations . . . 8
2.14 LU Decomposition, Cholesky and QRD Functions . . . 9
2.15 Special Linear System Solvers . . . 9
2.16 Controlling the Number of Threads . . . 9

3 Support Functions **11**

3.1 Initialization and Finalization 11
csipl_init 12
csipl_finalize 14
3.2 Sundry Functions 16
csipl_complete 17
csipl_cstorage 18

4 Scalar Functions **19**
4.1 Real Scalar Functions 19
csipl_acos_f 20
csipl_asin_f 21
csipl_atan_f 22
csipl_atan2_f 23
csipl_ceil_f 24
csipl_cos_f 25
csipl_cosh_f 26
csipl_exp_f 27
csipl_exp10_f 28
csipl_floor_f 29
csipl_fmod_f 30
csipl_hypot_f 31
csipl_log_f 32
csipl_log10_f 33
csipl_mag_f 34
csipl_max_P 35
csipl_min_P 36
csipl_pow_f 37
csipl_rsqrt_f 38
csipl_sin_f 39
csipl_sinh_f 40
csipl_sqrt_f 41
csipl_tan_f 42
csipl_tanh_f 43
4.2 Complex Scalar Functions 44

csipl_arg_f 45
csipl_cadd_f 46
csipl_rcadd_f 47
csipl_cdiv_f 48
csipl_crdiv_f 49
csipl_cexp_f 50
csipl_cjmul_f 51
csipl_cmag_f 52
csipl_cmagsq_f 53
csipl_cmplx_f 54
csipl_cmul_f 55
csipl_rcmul_f 56
csipl_cneg_f 57
csipl_conj_f 58
csipl_crecip_f 59
csipl_csqrt_f 60
csipl_csub_f 61
csipl_rcsub_f 62
csipl_crsub_f 63
csipl_imag_f 64
csipl_polar_f 65
csipl_real_f 66
csipl_rect_f 67
4.3 Index Scalar Functions 68
csipl_matindex 69
csipl_mcolindex 70
csipl_mrowindex 71

5 Random Number Generation **72**
5.1 Random Number Functions 72
csipl_randcreate 73
csipl_randdestroy 75
csipl_randu_f 76
csipl_crandu_f 77

csipl_vrandu_f 78
csipl_cvrandu_inter_f 79
csipl_cvrandu_split_f 80
csipl_randn_f 81
csipl_crandn_f 82
csipl_vrandn_f 83
csipl_cvrandn_inter_f 84
csipl_cvrandn_split_f 85

6 Vector And Elementwise Operations 86

6.1 Elementary Mathematical Functions 86
csipl_vacos_f 88
csipl_macos_f 89
csipl_vasin_f 90
csipl_masin_f 91
csipl_vatan_f 92
csipl_matan_f 93
csipl_vatan2_f 94
csipl_matan2_f 96
csipl_vcos_f 98
csipl_mcos_f 99
csipl_vcosh_f 100
csipl_mcosh_f 101
csipl_vexp_f 102
csipl_cvexp_inter_f 103
csipl_cvexp_split_f 104
csipl_mexp_f 106
csipl_cmexp_inter_f 107
csipl_cmexp_split_f 109
csipl_vexp10_f 111
csipl_mexp10_f 112
csipl_vfloor_f 113
csipl_vlog_f 114
csipl_cvlog_inter_f 115

csipl_cvlog_split_f 116
csipl_mlog_f 118
csipl_cmlog_inter_f 119
csipl_cmlog_split_f 120
csipl_vlog10_f 122
csipl_mlog10_f 123
csipl_vsin_f 124
csipl_msin_f 125
csipl_vsinh_f 126
csipl_msinh_f 127
csipl_vsqrt_f 128
csipl_cvsqrt_inter_f 129
csipl_cvsqrt_split_f 130
csipl_msqrt_f 131
csipl_cmsqrt_inter_f 132
csipl_cmsqrt_split_f 133
csipl_vtan_f 134
csipl_mtan_f 135
csipl_vtanh_f 136
csipl_mtanh_f 137
6.2 Unary Operations 138
csipl_varg_f 140
csipl_marg_f 141
csipl_vceil_f 142
csipl_cvconj_inter_f 143
csipl_cvconj_split_f 144
csipl_cmconj_inter_f 145
csipl_cmconj_split_f 146
csipl_vcumsum_P 147
csipl_cvcumsum_inter_P 148
csipl_cvcumsum_split_P 149
csipl_mcumsum_P 151
csipl_cmcumsum_inter_P 153

csipl_cmcumsum_split_P 154
csipl_veuler_f 156
csipl_meuler_f 157
csipl_vmag_P 158
csipl_cvmag_inter_f 159
csipl_cvmag_split_f 160
csipl_mmag_f 161
csipl_cmmag_inter_f 162
csipl_cmmag_split_f 163
csipl_vcmagsq_inter_f 164
csipl_vcmagsq_split_f 165
csipl_mcmagsq_f 166
csipl_vmeanval_f 167
csipl_cvmeanval_inter_f 168
csipl_cvmeanval_split_f 169
csipl_mmeanval_f 170
csipl_cmmeanval_inter_f 171
csipl_cmmeanval_split_f 172
csipl_vmeansqval_f 173
csipl_cvmeansqval_inter_f 174
csipl_cvmeansqval_split_f 175
csipl_mmeansqval_f 176
csipl_cmmeansqval_inter_f 177
csipl_cmmeansqval_split_f 178
csipl_vmodulate_f 179
csipl_cvmodulate_inter_f 181
csipl_cvmodulate_split_f 183
csipl_vneg_P 185
csipl_cvneg_inter_f 186
csipl_cvneg_split_f 187
csipl_mneg_P 188
csipl_cmneg_inter_f 189
csipl_cmneg_split_f 190

csipl_vrecip_f 191
csipl_cvrecip_inter_f 192
csipl_cvrecip_split_f 193
csipl_mrecip_f 194
csipl_cmrecip_inter_f 195
csipl_cmrecip_split_f 196
csipl_vrsqrt_f 198
csipl_mrsqrt_f 199
csipl_vsq_f 200
csipl_msq_f 201
csipl_vsumval_P 202
csipl_cvsumval_inter_P 203
csipl_cvsumval_split_P 204
csipl_msumval_P 205
csipl_cmsumval_inter_P 206
csipl_cmsumval_split_P 207
csipl_vsumval_bl 208
csipl_vsumsqval_f 209
csipl_msumsqval_f 210
6.3 Binary Operations 211
csipl_vadd_P 215
csipl_cvadd_inter_P 217
csipl_cvadd_split_P 219
csipl_madd_P 221
csipl_cmadd_inter_f 223
csipl_cmadd_split_f 224
csipl_rcvadd_inter_f 226
csipl_rcvadd_split_f 227
csipl_rcmadd_inter_f 229
csipl_rcmadd_split_f 230
csipl_svadd_P 232
csipl_csvadd_inter_f 233
csipl_csvadd_split_f 234

csipl_smadd_P 236
csipl_csmadd_inter_f 238
csipl_csmadd_split_f 239
csipl_rscvadd_inter_f 241
csipl_rscvadd_split_f 242
csipl_rscmadd_inter_f 243
csipl_rscmadd_split_f 244
csipl_Dvdiv_P 246
csipl_Dvdiv_inter_P 248
csipl_Dvdiv_split_P 250
csipl_mdiv_f 252
csipl_cmdiv_inter_f 254
csipl_cmdiv_split_f 256
csipl_rcvdiv_inter_f 258
csipl_rcvdiv_split_f 259
csipl_rcmdiv_inter_f 261
csipl_rcmdiv_split_f 262
csipl_crmdiv_inter_f 264
csipl_crmdiv_split_f 266
csipl_rscmsub_inter_f 268
csipl_rscmsub_split_f 269
csipl_vsdiv_f 271
csipl_cvrsdiv_inter_f 272
csipl_cvrsdiv_split_f 273
csipl_msdiv_f 275
csipl_cmrsdiv_inter_f 276
csipl_cmrsdiv_split_f 277
csipl_vexpoavg_f 279
csipl_cvexpoavg_inter_f 280
csipl_cvexpoavg_split_f 281
csipl_mexpoavg_f 282
csipl_cmexpoavg_inter_f 283
csipl_cmexpoavg_split_f 284

csipl_vhypot_f 286
csipl_mhypot_f 287
csipl_cvjmul_inter_f 288
csipl_cvjmul_split_f 289
csipl_cmjmul_inter_f 291
csipl_cmjmul_split_f 292
csipl_Dvmul_P 294
csipl_cvmul_inter_f 296
csipl_cvmul_split_f 297
csipl_mmul_P 299
csipl_cmmul_inter_f 301
csipl_cmmul_split_f 302
csipl_rcvmul_inter_f 304
csipl_rcvmul_split_f 305
csipl_rcmmul_inter_f 307
csipl_rcmmul_split_f 308
csipl_rscvmul_inter_f 310
csipl_rscvmul_split_f 311
csipl_csvmul_inter_f 312
csipl_csvmul_split_f 313
csipl_smmul_f 315
csipl_csmmul_inter_f 316
csipl_csmmul_split_f 317
csipl_rscmmul_inter_f 319
csipl_rscmmul_split_f 320
csipl_vmmul_f 322
csipl_cvmmul_inter_f 324
csipl_cvmmul_split_f 326
csipl_rvcmmul_inter_f 328
csipl_rvcmmul_split_f 330
csipl_vsub_P 332
csipl_cvsub_inter_f 334
csipl_cvsub_split_f 335

csipl_msub_P 337
csipl_cmsub_inter_f 339
csipl_cmsub_split_f 340
csipl_crmsub_inter_f 342
csipl_crmsub_split_f 343
csipl_rcvsub_inter_f 345
csipl_rcvsub_split_f 346
csipl_rcmsub_inter_f 348
csipl_rcmsub_split_f 349
csipl_crvsub_inter_f 351
csipl_crvsub_split_f 352
csipl_svsub_P 354
csipl_csvsub_inter_f 355
csipl_csvsub_split_f 356
csipl_smsub_P 358
csipl_csmsub_inter_f 360
csipl_csmsub_split_f 361
csipl_smdiv_f 363
csipl_csmdiv_inter_f 364
csipl_csmdiv_split_f 365
csipl_rscvdiv_inter_f 367
csipl_rscvdiv_split_f 368
csipl_csvdiv_inter_f 370
csipl_csvdiv_split_f 371
csipl_rscvsub_inter_f 373
csipl_rscvsub_split_f 374
csipl_rscmdiv_inter_f 375
csipl_rscmdiv_split_f 376
6.4 Ternary Operations 378
csipl_vam_P 379
csipl_cvam_inter_P 381
csipl_cvam_split_P 383
csipl_vmsa_f 385

csipl_cvmsa_inter_f 386
csipl_cvmsa_split_f 387
csipl_vmsb_f 389
csipl_cvmsb_inter_f 391
csipl_cvmsb_split_f 393
csipl_vsam_f 395
csipl_cvsam_inter_f 396
csipl_cvsam_split_f 397
csipl_vsbm_f 399
csipl_cvsbm_inter_f 401
csipl_cvsbm_split_f 403
csipl_vsma_f 405
csipl_cvsma_inter_f 406
csipl_cvsma_split_f 407
csipl_vsmsa_f 409
csipl_cvsmsa_inter_f 410
csipl_cvsmsa_split_f 411
6.5 Logical Operations 413
csipl_valltrue_bl 414
csipl_malltrue_bl 415
csipl_vanytrue_bl 416
csipl_manytrue_bl 417
csipl_vleq_P 418
csipl_cvleq_inter_P 420
csipl_cvleq_split_P 422
csipl_mleq_P 424
csipl_cmleq_inter_P 426
csipl_cmleq_split_P 428
csipl_vlge_P 430
csipl_mlge_P 432
csipl_vlgt_P 434
csipl_mlgt_P 436
csipl_vlle_P 438

csipl_mlle_P 440
csipl_vllt_P 442
csipl_mllt_P 444
csipl_vlne_P 446
csipl_cvlne_inter_P 448
csipl_cvlne_split_P 450
csipl_mlne_P 452
csipl_cmlne_inter_P 454
csipl_cmlne_split_P 456
6.6 Selection Operations 458
csipl_vclip_P 459
csipl_mclip_f 461
csipl_vinvclip_P 463
csipl_minvclip_f 465
csipl_vindexbool 467
csipl_vmax_f 468
csipl_mmax_f 469
csipl_vmaxmg_f 470
csipl_mmaxmg_f 471
csipl_vcmaxmgsq_inter_f 472
csipl_vcmaxmgsq_split_f 473
csipl_vcmaxmgsqval_inter_f 475
csipl_vcmaxmgsqval_split_f 476
csipl_vmaxmgval_f 477
csipl_mmaxmgval_f 478
csipl_vmaxval_f 479
csipl_mmaxval_f 480
csipl_vmin_f 481
csipl_mmin_f 482
csipl_vminmg_f 483
csipl_mminmg_f 484
csipl_vcminmgsq_inter_f 485
csipl_vcminmgsq_split_f 486

csipl_vcminmgsqval_inter_f 488
csipl_vcminmgsqval_split_f 489
csipl_vminmgval_f 490
csipl_mminmgval_f 491
csipl_vminval_f 492
csipl_mminval_f 493
6.7 Bitwise and Boolean Logical Operators 494
csipl_vand_P 495
csipl_mand_P 497
csipl_vnot_P 499
csipl_mnot_P 500
csipl_vor_P 501
csipl_mor_P 503
csipl_vxor_P 505
csipl_mxor_P 507
6.8 Element Generation and Copy 509
csipl_vcopy_P_P 510
csipl_cvcopy_inter_f_f 512
csipl_cvcopy_split_f_f 513
csipl_mcopy_P_P 515
csipl_cmcopy_inter_f_f 517
csipl_cmcopy_split_f_f 518
csipl_vfill_P 520
csipl_cvfill_inter_f 521
csipl_cvfill_split_f 522
csipl_mfill_P 523
csipl_cmfill_inter_f 524
csipl_cmfill_split_f 525
csipl_vramp_P 526
6.9 Manipulation Operations 527
csipl_vcmplx_inter_f 529
csipl_vcmplx_split_f 530
csipl_mcmplx_f 531

csipl_vgather_P 532
csipl_cvgather_inter_f 534
csipl_cvgather_split_f 536
csipl_mgather_P 538
csipl_cmgather_inter_f 540
csipl_cmgather_split_f 542
csipl_vimag_inter_f 544
csipl_vimag_split_f 545
csipl_mimag_f 546
csipl_vpolar_inter_f 547
csipl_vpolar_split_f 548
csipl_mpolar_inter_f 550
csipl_mpolar_split_f 552
csipl_vreal_inter_f 554
csipl_vreal_split_f 555
csipl_mreal_f 556
csipl_vrect_inter_f 557
csipl_vrect_split_f 558
csipl_mrect_inter_f 560
csipl_mrect_split_f 562
csipl_vscatter_P 564
csipl_cvscatter_inter_f 566
csipl_cvscatter_split_f 568
csipl_mscatter_P 570
csipl_cmscatter_inter_f 572
csipl_cmscatter_split_f 574
csipl_vswap_P 576
csipl_cvswap_inter_f 578
csipl_cvswap_split_f 579
csipl_mswap_P 581
csipl_cmswap_inter_f 583
csipl_cmswap_split_f 584

7 Signal Processing Functions **586**

7.1 FFT Functions 586
csipl_ccfftip_create_f 588
csipl_ccfftop_create_f 590
csipl_crfftop_create_f 592
csipl_rcfftop_create_f 594
csipl_ccfftip_inter_f 596
csipl_ccfftip_split_f 597
csipl_ccfftop_inter_f 599
csipl_ccfftop_split_f 601
csipl_crfftop_inter_f 603
csipl_crfftop_split_f 605
csipl_rcfftop_inter_f 607
csipl_rcfftop_split_f 609
csipl_fft_destroy_f 611
csipl_fft_getattr_f 612
csipl_ccfftmop_create_f 613
csipl_ccfftmip_create_f 615
csipl_crfftmop_create_f 617
csipl_rcfftmop_create_f 619
csipl_fftm_destroy_f 621
csipl_fftm_getattr_f 622
csipl_ccfftmip_inter_f 623
csipl_ccfftmip_split_f 625
csipl_ccfftmop_inter_f 627
csipl_ccfftmop_split_f 629
csipl_crfftmop_inter_f 631
csipl_crfftmop_split_f 633
csipl_rcfftmop_inter_f 635
csipl_rcfftmop_split_f 637
csipl_ccfft2dop_create_f 639
csipl_ccfft2dip_create_f 641
csipl_crfft2dop_create_f 643
csipl_rcfft2dop_create_f 645

csipl_ccfft2dip_inter_f 647
csipl_ccfft2dip_split_f 648
csipl_ccfft2dop_inter_f 650
csipl_ccfft2dop_split_f 652
csipl_crfft2dop_inter_f 654
csipl_crfft2dop_split_f 656
csipl_rcfft2dop_inter_f 658
csipl_rcfft2dop_split_f 660
csipl_fft2d_destroy_f 662
csipl_fft2d_getattr_f 663
7.2 Convolution/Correlation Functions 664
csipl_conv1d_create_f 665
csipl_conv1d_destroy_f 667
csipl_conv1d_getattr_f 668
csipl_convolve1d_f 669
csipl_conv2d_create_f 671
csipl_conv2d_destroy_f 674
csipl_conv2d_getattr_f 675
csipl_convolve2d_f 676
csipl_Dcorr1d_create_P 678
csipl_Dcorr1d_destroy_P 681
csipl_Dcorr1d_getattr_P 682
csipl_correlate1d_f 684
csipl_ccorrelate1d_inter_f 686
csipl_ccorrelate1d_split_f 688
csipl_Dcorr2d_create_P 690
csipl_Dcorr2d_destroy_P 693
csipl_Dcorr2d_getattr_P 694
csipl_correlate2d_f 696
csipl_ccorrelate2d_inter_f 698
csipl_ccorrelate2d_split_f 700
7.3 Window Functions 702
csipl_vcreate_blackman_f 703

csipl_vcreate_cheby_f 705
csipl_vcreate_hanning_f 707
csipl_vcreate_kaiser_f 709
7.4 Filter Functions 711
csipl_fir_create_f 712
csipl_cfir_create_inter_f 714
csipl_cfir_create_split_f 716
csipl_Dfir_destroy_P 718
csipl_firflt_f 719
csipl_cfirflt_inter_f 721
csipl_cfirflt_split_f 723
csipl_Dfir_getattr_P 725
csipl_Dfir_reset_P 727
7.5 Miscellaneous Signal Processing Functions 728
csipl_vhisto_f 729

8 Linear Algebra **731**
8.1 Matrix and Vector Operations 731
csipl_cmherm_inter_f 733
csipl_cmherm_split_f 734
csipl_cvjdot_inter_f 736
csipl_cvjdot_split_f 737
csipl_gemp_f 738
csipl_cgemp_inter_f 740
csipl_cgemp_split_f 742
csipl_gems_f 744
csipl_cgems_inter_f 746
csipl_cgems_split_f 748
csipl_mprod_P 750
csipl_cmprod_inter_P 752
csipl_cmprod_split_P 754
csipl_cmprodh_inter_P 756
csipl_cmprodh_split_P 758
csipl_cmprodj_inter_P 760

csipl_cmprodj_split_P 762
csipl_mprodt_P 764
csipl_cmprodt_inter_P 766
csipl_cmprodt_split_P 768
csipl_mvprod_P 770
csipl_cmvprod_inter_P 772
csipl_cmvprod_split_P 774
csipl_mtrans_P 776
csipl_cmtrans_inter_P 778
csipl_cmtrans_split_P 780
csipl_vdot_f 782
csipl_cvdot_inter_f 783
csipl_cvdot_split_f 784
csipl_vmprod_P 785
csipl_cvmprod_inter_P 787
csipl_cvmprod_split_P 789
csipl_vouter_f 791
csipl_cvouter_inter_f 793
csipl_cvouter_split_f 795
csipl_vcsummgval_inter_f 797
csipl_vcsummgval_split_f 798
csipl_minvlu_f 799
csipl_cminvlu_inter_f 800
csipl_cminvlu_split_f 801
8.2 Special Linear System Solvers 803
csipl_covsol_f 804
csipl_ccovsol_inter_f 806
csipl_ccovsol_split_f 808
csipl_llsqsol_f 810
csipl_cllsqsol_inter_f 812
csipl_cllsqsol_split_f 814
csipl_toepsol_f 816
csipl_ctoepsol_inter_f 818

csipl_ctoepsol_split_f 820
8.3 General Square Linear System Solver 822
csipl_lud_f 823
csipl_clud_inter_f 824
csipl_clud_split_f 825
csipl_Dlud_create_P 826
csipl_Dlud_destroy_P 827
csipl_Dlud_getattr_P 828
csipl_lusol_f 829
csipl_clusol_inter_f 831
csipl_clusol_split_f 833
8.4 Symmetric Positive Definite Linear System Solver 835
csipl_chold_f 836
csipl_cchold_inter_f 838
csipl_cchold_split_f 840
csipl_chold_create_f 842
csipl_cchold_create_f 844
csipl_Dchold_destroy_P 846
csipl_Dchold_getattr_P 847
csipl_cholsol_f 848
csipl_ccholsol_inter_f 849
csipl_ccholsol_split_f 850
8.5 Overdetermined Linear System Solver 851
csipl_qrd_f 852
csipl_cqrd_inter_f 853
csipl_cqrd_split_f 854
csipl_qrd_create_f 856
csipl_cqrd_create_f 858
csipl_Dqrd_destroy_P 860
csipl_Dqrd_getattr_P 861
csipl_qrdprodq_f 862
csipl_cqrdprodq_inter_f 864
csipl_cqrdprodq_split_f 866

csipl_qrdsolr_f 868
csipl_cqrdsolr_inter_f 870
csipl_cqrdsolr_split_f 872
csipl_qrsol_f 874
csipl_cqrsol_inter_f 876
csipl_cqrsol_split_f 878

Chapter 1. Introduction

This document describes the functions available in the CSIPL library of vector, signal processing, and linear algebra routines.

1.1 Types

The following base types are available:

`csipl_bool`	boolean
`csipl_cscalar_f`	complex floating-point scalar
`csipl_index`	index to a vector
`csipl_index_mi`	index (pair of integers) to a matrix
`csipl_length`	dimension of a vector or matrix
`csipl_stride`	stride or jump of a vector or matrix

If you are using a version of the library that has 64-bit pointers, you must define the symbol `POINTER_SIZE_64BIT` at compile time: this will ensure that variables of types `csipl_length` and `csipl_stride` are 64-bit integers.

The library also passes information around in abstract data types. These objects are opaque structures (implemented as incomplete typedefs) and they can only be created, accessed, and destroyed with library functions that reference them via a pointer. Some are used to describe the data layout in memory; others store information on filters, matrix decompositions, and so on. Some objects have a 'get attribute' function that allows the user access to the internal values.

1.2 Symbols and Flags

The following symbolic constants are defined.

```
CSIPL_TRUE
CSIPL_FALSE
```

Other symbols are defined in enumerated types. The valid choices are listed with each function description.

1.3 Vector Variables

A vector is passed into a function with three parameters:

- a pointer to the start of the data
- an integer for the distance between processed elements (stride); stride = 1 (unit stride) means every element of a vector has to be processed, stride = 2 means every other element, and so on
- an integer for the number of elements accessed (length).

Stride and length are measured in number of data elements (not bytes). For a compound type like complex, this means the number of (real,imaginary) pairs and not the number of atomic type elements (floats for example).

When several vectors are passed into a function, each has its own stride, but usually they all share a common length. In the function prototypes, each vector variable has two parameters (a data pointer followed by a stride) and the common processed length is usually the last parameter.

When a stride is negative the processing progresses backwards through the data so a suitable offset must be added to the data pointer. With proper pointer arithmetic the offset is also measured in number of data elements.

1.4 Matrix Variables

Matrices are stored in row-major format. A matrix is passed into a function with four parameters:

- a pointer to the start of the data
- an integer for the distance between adjacent elements in a column of the matrix (leading dimension); it is denoted by `ldM` for matrix `M`; it can be viewed as the number of columns in the 2-dimensional array containing `M`;
- an integer for the number of rows
- an integer for the number of columns (less than or equal to the leading dimension).

As with vectors, the leading dimension is measured in number of data elements.

When several matrices are passed into a function, each has its own leading dimension, but often they all have the same dimensions, or have one dimension in common. In the function prototypes, each matrix variable has two parameters (a data pointer followed by a leading dimension); the dimension parameters (numbers of row and columns) are often common for several matrices and are usually the last items in the parameter list. The description of each function explains the dimensions of the matrices.

1.5 Complex Variables

Most functions that use complex data are available in two versions: one accepts data stored in interleaved format and each complex variable is passed as a single parameter; the other takes data stored in split format and each complex variable is passed via two parameters — one each for the real and imaginary parts.

1.6 Function Names

For almost all functions, the suffix indicates the base datatype of the arguments as follows:

`f`	float
`i`	integer
`bl`	boolean
`vi`	vector index
`mi`	matrix index.

1.7 Hints

They following mechanisms are provided for the programmer to indicate preferences for optimisation: currently **almost all are ignored** but they are reserved for future use. The only exception is the algorithm hint in FIR filters.

- Flags of the enumerated type `csipl_memory_hint` specified when allocating or creating some objects.
- Flags of the enumerated type `csipl_alg_hint` used to indicate whether algorithmic optimisation should minimise execution time, memory use, or maximise numerical accuracy.
- An indication of how many times an object will be used (filters and FFTs have such a parameter).

1.8 Notation

The following standard mathematical notation is used in the function descriptions.

$:=$	assignment operator
i	square root of -1
$\lvert x \rvert$	absolute value of the real number x
$\lvert z \rvert$	modulus of the complex number z
$\lfloor x \rfloor$	floor of the real number x (largest integer less than or equal to x)
$\lceil x \rceil$	ceiling of the real number x (smallest integer greater than or equal to x)
z^*	conjugate of the complex number z
M^{T}	transpose of the matrix M
M^{H}	Hermitian (conjugate transpose) of the complex matrix M

Note that in expressions i is always the square root of -1; vectors and matrices are indexed with j and k.

An elementwise operation on vectors will be written $C[j * \mathrm{strideC}] := A[j * \mathrm{strideA}] + B[j * \mathrm{strideB}]$. Often the range of the index variable is not given explicity; in such cases it is clear from the context that it runs over all the elements in the vectors and that the lengths of the vectors must be equal.

An M by N matrix has M rows and N columns.

1.9 Errors and Restrictions

Many functions require that their arguments be **conformant**. This means that the objects passed have compatible attributes: for example, size and shape of matrices, lengths of vectors or filter kernels.

If an argument is required to be **valid**, it means:

- a pointer is not `NULL`

- a flag is a member of the required enumerated type
- an object has been initialised and not destroyed.

Errors can occur for the following reasons:

1. argument is outside the domain for calculation
2. over/underflow during calculation
3. failure to allocate memory
4. algorithm failure because of inappropriate data (as when a matrix does not have full rank)
5. arguments are invalid, out of range, or non-conformant.

Only errors of type 5 are regarded as fatal: in this case, the development version of the library will write a message to `stderr` and call `exit`.

Errors of types 3 and 4 are signalled through the return value of the function. A create function will return `NULL` if the allocation fails; functions with integer return codes use zero to indicate success.

The calling program is not alerted to errors of types 1 and 2.

Chapter 2. Getting the Best Performance

This section is a short guide for programmers using the NAS CSIPL Library. It contains explanations of library behavior, and tips on selecting the right storage options for your data to increase performance.

2.1 Version Information

Information about the version of the NAS CSIPL library you are using can be found in the comments at the top of the include file `csipl.h`. There is no way that a program can determine the library version at run-time.

2.2 Memory Alignment

The efficiency of many operations is improved if data within memory is correctly aligned on certain word boundaries.

Vectors and matrices can be loaded and stored faster if they are vector aligned.

The following table gives the vector alignment and minimum vector length for float data:

Technology	Vector Aligment
SSE	16
AVX	32
AltiVec	16

Alignment can be controlled using a function such as `memalign`. This is a C function that is not in the ANSI standard but is available on many systems. It is defined in `malloc.h` on Linux systems.

The following macro redefines `malloc` so that all memory allocation is optimally aligned:

```
#include <malloc.h>
#define malloc(SIZE) memalign(16, SIZE)
```

Some operating systems (*e.g.* Apple's OSX) automatically align all memory to a 16-byte boundary so `memalign` is not needed.

2.3 Vector/Matrix Format

When available, vector and matrix calculations are done using single instruction, multiple data (SIMD) instructions to process several elements simultaneously. This imposes a minimum vector length given in the table below:

Technology Vector Length (floats)	minimum
SSE	4
AVX	8
AltiVec	4

For short ints (16 bits) the vector length should be twice that of the float vector length. If the vector unit supports doubles (64 bits), then the vector length should be half that of floats.

For best performance all input and output vectors should:

- have a stride of 1

  Vectors and matrices can be loaded and stored much quicker when they are contiguous in memory. The library includes special optimisations for a stride length of 2 (which was added for interleaved complex numbers), but all other non-unit strides will be significantly slower than a stride of 1 and, in many cases, almost as slow as unvectorised scalar code. Note that, a stride of -1 will also be significantly slower than a stride of $+1$.

- be vector aligned

- have length greater than or equal to the vector length

  The vector unit works on arrays of the vector length so no speed up is gained by using the library on vectors of length less than this.

- have row (row major matrices) or column (column major matrices) length divisible by the vector length

  For a row major matrix: if the row length of a matrix is not divisible by the vector length then the alignment of the first element of each row will change for each row/column. For optimal performance the first element of each row should be vector aligned.

  The same rule applies to columns in column major matrices.

- have a length divisible by the vector length

  Any elements at the end of the vector which cannot be dealt with by the vector unit must be dealt with in normal scalar code, which will decrease the performance. The decrease in performance becomes less important for longer vectors.

2.4 Error Checking and Debugging

Two versions of the NAS CSIPL library are provided: a performance version and a development version. The development version of the library (signified by a 'D' in the library's name) contains full error checking and should always be used when developing and debugging applications.

A few library functions return status information: always check the return code of those that do.

The performance version of the library contains no error checking, and consequently runs faster than the development library. The performance library should only be used with applications that have been run successfully with the development version of the library.

When timing code, the performance version of the library should be used.

Note: the performance version of the library reads in data before it knows how much will be used and as a result often reads more data than is needed. This is not a problem, except when using memory checkers such as Electric Fence which object to this behavior. The development library only reads in the data it intends to use and so is safe to use with memory checkers.

2.5 Support Functions

Always call `csip_init` and `csip_finalize` at the beginning and end of a program.

Note: For the AltiVec optimized library, calling `csip_init` will put the AltiVec unit into non-Java mode if it is not already. This speeds up most AltiVec instructions.

2.6 Scalar Functions

As the vector unit works on arrays of the vector length, scalar functions in the library are not vectorized.

2.7 Random Number Generation

The random number generation functions have not been vectorized in the current version of the library.

2.8 Vector and Elementwise Operations

All vector and elementwise operations work optimally on vectors which match the conditions given in Section 2.3.

2.9 Signal Processing Functions

All signal processing operations work optimally on vectors which match the conditions given in Section 2.3.

Most of the signal processing routines are split into three stages:

- a create stage
- a compute stage
- a destroy stage.

The library has been optimized to minimize the time taken to do the compute stage, which means as much precomputation as possible is done in the create stage. If you are using the same signal processing routine on many vectors of the same length, it is far

quicker to just create the required signal processing object once and reuse it for each computation stage rather than recreating the object each time it is needed.

2.10 FFT Functions

To get the best performance from an FFT, a vector must have length a multiple of the numbers 2, 4, 8, and 3 only. If a vector length is not a multiple of these numbers, a DFT may be done, which is considerably slower than an FFT. Factors of 3 should be avoided if possible. An FFT will only be done for factors of 3 if the length also has a factor of 16, otherwise a DFT is done.

When doing large FFT's, optimal routines have been developed for the lengths: 4096, 8192, 16384, 32768, and 65536. These lengths should be much quicker than lengths of similar magnitude. In-place FFT's are normally faster than out-of-place FFT's. FFT's are fastest with a scale factor of 1. However, if you need to use a different scale factor, it is better to let the FFT routine do the scaling rather than to do it yourself.

The internal FFT routines only work on vector aligned data with a stride of 1. If vectors are used which do not match these restrictions an internal copy of the vector will be made. This is an important consideration when using large vectors. Also, if complex vectors are not stored split an internal copy will be made.

The current version of the NAS CSIPL library does not have any special FFT routines for doing multiple FFT's, so the time to do n single FFT's will be approximately the same as using the multiple FFT routines on a matrix of n rows.

The `ntimes` parameter to the FFT functions is ignored. The algorithmic hint is only used in the FFT create function: if the `CSIP_ALG_NOISE` hint is used, the FFT create function will take significantly longer. By default, the algorithms are optimized to minimize execution time.

2.11 FIR Filter, Convolution and Correlation Functions

These functions call the FFT functions internally and are therefore subject to the same restrictions.

Hints are ignored with the exception of the internal calling of the FFT create function described in the FFT functions section.

2.12 Linear Algebra Functions

For optimal performance the vectors and matrices used with the linear algebra functions should match the conditions given in Section 2.3. (See also the sections below when using complex LU, complex Cholesky, or complex QRD functions).

2.13 Matrix and Vector Operations

Matrix and vector operations should work optimally on row or column major matrices (row major is the default), however, the restriction exists that all matrices passed to a function should be of the same order. For example, using two row major matrices as input to a function and a column major as output will be slower than using all row major or all column major. When matrices are passed to NAS CSIPL functions that

are not all of the same order, the library will assume they are all row major and treat the column major matrices as strided matrices. (See also the sections below when using LU, Cholesky, or QRD functions).

2.14 LU Decomposition, Cholesky and QRD Functions

These functions have three separate stages:

- a create stage
- a compute stage
- a destroy stage.

The library has been optimized to minimize the time taken to do the compute stage, which means as much precomputation as possible is done in the create stage. If you are using the linear algebra routine on many matrices of the same size, it is far quicker to just create the required linear algebra object once and reuse it for each computation stage rather than recreating the object each time it is needed.

If matrices of different orders or strided matrices are passed to these functions, an internal copy will be made of the entire matrix before the computation is done. This is an important consideration when using large matrices. (Note: unaligned matrices do NOT require internal copying provided they have a stride of one and all matrices used with the functions are of the same order).

When using the QRD functions, it is only necessary to save the Q matrix if using the `qrdprodq` function; the `qrsol` and `qrdsolr` do not need the Q matrix.

2.15 Special Linear System Solvers

The `covsol` and `llsqsol` functions internally use the QRD functions and so have the same requirements for optimal performance.

The `toepsol` functions are based on vector operations and so have the same requirements for optimal performance.

2.16 Controlling the Number of Threads

The NAS CSIPL library is multithreaded and will take advantage of multiple cores on the processor invoking it. Utilizing multiple threads is automatic:

- The maximum number of threads used is set when `csip_init` is called.
- The maximum number running at any one time is also set at that point. If a threaded routine is called with (say) 4 threads and we have hit this maximum number running then four of them are shut down before the new function is executed.
- The number of threads invoked when a routine is called, is decided by that routine by reference to the data (vector or matrix) size specified in the call, to provide the best performance for that call.

It is possible to change the maximum number of threads used.

1. A threaded, and a non-threaded ("serial") version of the library are provided. If you wish to only ever use one thread in a library call, use the serial version of the library.

2. The maximum number of threads used for a specific function call, and the maximum number kept running at any one time, can be changed by a call to the routine `Thread_SetParams` with arguments `num_threads` and `max_num_running`. This call, if used, *must* be made before the library initialization routine `csip_init` is called.

3. When calling the routine `Thread_SetParams` the value of `max_num_running` must be greater or equal to 3*`num_threads`. If the user enters a smaller value than this in their `Thread_SetParams` function call then the function will set the value of `max_num_running` to 3*`num_threads`.

   If no call to `Thread_SetParams` is made, the library default values will be utilized.

Chapter 3. Support Functions

3.1 Initialization and Finalization

- `csipl_init`
- `csipl_finalize`

csipl_init

Provides initialization, allowing the library to allocate and set a global state, and prepare to support the use of CSIPL functionality by the user.

Prototype

```
int csipl_init(
                    void *ptr);
```

Parameters

- `ptr`, pointer to structure, input.

Return Value

- Error code.

Description

This required function informs the CSIPL library that library initialization is requested, and that other CSIPL functions will be called. This function must be called at least once by all CSIPL programs. It may be called multiple times as well, with corresponding calls to `csipl_finalize` to create nested pairs of initialization/termination. Only the `csipl_finalize` matching the first `csipl_init` call will actually release the library. Intermediate calls to `csipl_init` have no effect, but support easy program/library development through compositional programming, where the user may not even know that a library itself invokes CSIPL.

The argument is reserved for future purposes. The `NULL` pointer should be passed for CSIPL 1.0 compliance.

Returns zero if the initialization succeeded, and non-zero otherwise.

Restrictions

This function may be called at any time during the execution of the program.

Errors

Notes

All programs must use the initialization function before calling any other CSIPL functions.

Unsuccessful initialization of the library is not an error. It is always signalled via the function's return value, and should always be checked by the application.

csipl_finalize

Provides cleanup and releases resources used by CSIPL (if the last of a nested series of calls), allowing the library to guarantee that any resources allocated by `csipl_init` are no longer in use after the call is complete.

Prototype

```
int csipl_finalize(
                    void *ptr);
```

Parameters

- `ptr`, pointer to structure, input.

Return Value

- Error code.

Description

This required function informs the CSIPL library that it is no longer being used by a program, so that all needed global state and hardware state can be returned. All programs must call this function at least once if they terminate. If the program does terminate, the last CSIPL function called must be an outermost `csipl_finalize`. Because nested `csipl_init`'s are supported, so are nested `csipl_finalize`'s.

The user must explicitly destroy all CSIPL objects before calling this function if this is an 'outermost' `csipl_finalize`. When nesting initializations, there is no need to destroy all objects prior to calling this function, but the user is obliged to keep track of the nesting depth if programs are written in such a manner.

Returns zero if the finalization succeeded, and non-zero otherwise. Zero is always returned if the call is not outermost.

Restrictions

This function may only be called if a previous `csipl_init` call has been made, with no previous corresponding `csipl_finalize`.

Errors

An outermost `csipl_finalize` call produces an error if there are any CSIPL objects not destroyed.

Notes

The user program is always responsible for returning resources it is no longer using by destroying CSIPL objects. An outermost finalization function will return resources that it allocated previously with `csipl_init`. Non-outermost `csipl_finalize`'s always return zero (success).

3.2 Sundry Functions

- `csipl_complete`
- `csipl_cstorage`

csipl_complete

Force all deferred CSIPL execution to complete.

Prototype

```
void csipl_complete(void);
```

Parameters

- none.

Return Value

- none.

Description

Forces all deferred CSIPL execution to complete and then returns. NOTE: there is no deferred execution in this implementation of CSIPL.

Restrictions

Errors

Notes

csipl_cstorage

Returns the preferred complex storage format for the system.

Prototype

```
csipl_cmplx_mem csipl_cstorage(void);
```

Parameters

- none.

Return Value

- enumerated type.

CSIPL_CMPLX_INTERLEAVED	interleaved
CSIPL_CMPLX_SPLIT	split
CSIPL_CMPLX_NONE	no preference

Description

Returns the preferred complex storage format for the system.

Restrictions

Errors

Notes

Chapter 4. Scalar Functions

4.1 Real Scalar Functions

- csipl_acos_f
- csipl_asin_f
- csipl_atan_f
- csipl_atan2_f
- csipl_ceil_f
- csipl_cos_f
- csipl_cosh_f
- csipl_exp_f
- csipl_exp10_f
- csipl_floor_f
- csipl_fmod_f
- csipl_hypot_f
- csipl_log_f
- csipl_log10_f
- csipl_mag_f
- csipl_max_P
- csipl_min_P
- csipl_pow_f
- csipl_rsqrt_f
- csipl_sin_f
- csipl_sinh_f
- csipl_sqrt_f
- csipl_tan_f
- csipl_tanh_f

csipl_acos_f

Computes the principal radian value in $[0, \pi]$ of the inverse cosine of a scalar.

Prototype

```
float  csipl_acos_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value $:= \cos^{-1}(\mathtt{A})$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

csipl_asin_f

Computes the principal radian value in $[0, \pi]$ of the inverse sine of a scalar.

Prototype

```
float  csipl_asin_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value $:= \sin^{-1}(\mathtt{A})$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

csipl_atan_f

Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent of a scalar.

Prototype

```
float  csipl_atan_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := $\tan^{-1}$(A).

Restrictions

Errors

Notes

csipl_atan2_f

Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of two scalars.

Prototype

```
float  csipl_atan2_f(
                    float  A,
                    float  B);
```

Parameters

- A, real scalar, input.
- B, real scalar, input.

Return Value

- real scalar.

Description

return value := $\tan^{-1}(\mathtt{A}/\mathtt{B})$.

The rules for calculating the function value are the same as those for the ANSI C function `atan2`.

Restrictions

The arguments must not be both zero.

Errors

Notes

csipl_ceil_f

Computes the ceiling of a scalar.

Prototype

```
float  csipl_ceil_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := $\lceil A \rceil$.

Returns the smallest integer greater than or equal to the argument.

Restrictions

Errors

Notes

csipl_cos_f

Computes the cosine of a scalar angle in radians.

Prototype

```
float  csipl_cos_f(
                   float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := cos(A).

Restrictions

Errors

Notes

Input argument is expressed in radians.

csipl_cosh_f

Computes the hyperbolic cosine of a scalar.

Prototype

```
float  csipl_cosh_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := cosh(A).

Restrictions

Errors

Notes

csipl_exp_f

Computes the exponential of a scalar.

Prototype

```
float  csipl_exp_f(
                   float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := exp(A).

Restrictions

Overflow will occur if the argument is greater than the natural logarithm of the maximum representable number. Underflow will occur if the argument is less than the natural logarithm of the maximum representable number.

Errors

Notes

csipl_exp10_f

Computes the base-10 exponential of a scalar.

Prototype

```
csipl_Dscalar_f csipl_exp10_f(
                    csipl_Dscalar_f A);
```

Parameters

- A, real or complex scalar, input.

Return Value

- real or complex scalar.

Description

return value := 10^{A}.

Restrictions

Overflow will occur if the argument is greater than the base-10 logarithm of the maximum representable number. Underflow will occur if the argument is less than the base-10 logarithm of the maximum representable number.

Errors

Notes

csipl_floor_f

Computes the floor of a scalar.

Prototype

```
float  csipl_floor_f(
                     float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := $\lfloor A \rfloor$.

Returns the largest integer less than or equal to the argument.

Restrictions

Errors

Notes

csipl_fmod_f

Computes the remainder of the quotient (modulo) of two scalars.

Prototype

```
float  csipl_fmod_f(
                    float  A,
                    float  B);
```

Parameters

- A, real scalar, input.
- B, real scalar, input.

Return Value

- real scalar.

Description

return value := A − nB

For some integer n such that, if B is non-zero, the result has the same sign as A and magnitude less than the magnitude of B.

Restrictions

Errors

Notes

csipl_hypot_f

Computes the square root of the sum of the squares (hypotenuse) of two scalars.

Prototype

```
float  csipl_hypot_f(
                     float  A,
                     float  B);
```

Parameters

- A, real scalar, input.
- B, real scalar, input.

Return Value

- real scalar.

Description

return value $:= \sqrt{\mathtt{A}^2 + \mathtt{B}^2}$.

Restrictions

Errors

Notes

csipl_log_f

Computes the natural logarithm of a scalar.

Prototype

```
float  csipl_log_f(
                   float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := $\log_e$(A).

Restrictions

The agument must be greater than zero.

Errors

Notes

csipl_log10_f

Computes the base 10 logarithm of a scalar.

Prototype

```
float  csipl_log10_f(
                     float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := $\log_{10}(A)$.

Restrictions

The argument must be greater than zero.

Errors

Notes

csipl_mag_f

Computes the magnitude (absolute value) of a scalar.

Prototype

```
float  csipl_mag_f(
                   float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := |A|.

Restrictions

Errors

Notes

csipl_max_P

Computes the maximum of two scalars.

Prototype

```
scalar_P  csipl_max_P(
                    scalar_P  A,
                    scalar_P  B);
```

The following instances are supported:

- `csipl_max_f`
- `csipl_max_i`

Parameters

- `A`, scalar, input.
- `B`, scalar, input.

Return Value

- scalar.

Description

return value := max{`A`, `B`}

Restrictions

Errors

Notes

csipl_min_P

Computes the minimum of two scalars.

Prototype

```
scalar_P  csipl_min_P(
                    scalar_P  A,
                    scalar_P  B);
```

The following instances are supported:

- csipl_min_f
- csipl_min_i

Parameters

- A, scalar, input.
- B, scalar, input.

Return Value

- scalar.

Description

return value := min{A, B}

Restrictions

Errors

Notes

csipl_pow_f

Computes the power function of two scalars.

Prototype

```
float  csipl_pow_f(
                    float  A,
                    float  B);
```

Parameters

- A, real scalar, input.
- B, real scalar, input.

Return Value

- real scalar.

Description

return value := A^B.

Restrictions

Errors

Notes

csipl_rsqrt_f

Computes the reciprocal square root of a scalar.

Prototype

```
float  csipl_rsqrt_f(
                     float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := $1/\sqrt{A}$.

Restrictions

The argument must be greater than or equal to zero.

Errors

Notes

csipl_sin_f

Computes the sine of a scalar angle in radians.

Prototype

```
float  csipl_sin_f(
                   float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := sin(A).

Restrictions

Errors

Notes

Input argument is expressed in radians.

csipl_sinh_f

Computes the hyperbolic sine of a scalar.

Prototype

```
float  csipl_sinh_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := sinh(A).

Restrictions

Errors

Notes

csipl_sqrt_f

Computes the square root of a scalar.

Prototype

```
float  csipl_sqrt_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := $\sqrt{A}$.

Restrictions

The argument must be greater than or equal to zero.

Errors

Notes

csipl_tan_f

Computes the tangent of a scalar angle in radians.

Prototype

```
float  csipl_tan_f(
                   float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := tan(A).

Restrictions

For values $(n + 1/2)\pi$, the tangent function has a singularity and is undefined.

Errors

Notes

Input argument is expressed in radians.

csipl_tanh_f

Computes the hyperbolic tangent of a scalar.

Prototype

```
float  csipl_tanh_f(
                    float  A);
```

Parameters

- A, real scalar, input.

Return Value

- real scalar.

Description

return value := tanh(A).

Restrictions

Errors

Notes

4.2 Complex Scalar Functions

- csipl_arg_f
- csipl_cadd_f
- csipl_rcadd_f
- csipl_cdiv_f
- csipl_crdiv_f
- csipl_cexp_f
- csipl_cjmul_f
- csipl_cmag_f
- csipl_cmagsq_f
- csipl_cmplx_f
- csipl_cmul_f
- csipl_rcmul_f
- csipl_cneg_f
- csipl_conj_f
- csipl_crecip_f
- csipl_csqrt_f
- csipl_csub_f
- csipl_rcsub_f
- csipl_crsub_f
- csipl_imag_f
- csipl_polar_f
- csipl_real_f
- csipl_rect_f

csipl_arg_f

Returns the argument in radians $[-\pi, \pi]$ of a complex scalar.

Prototype

```
float  csipl_arg_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- real scalar.

Description

return value := $\tan^{-1}(\text{imag}(x)/\text{real}(x))$.

Restrictions

The argument must not be zero.

Errors

Notes

csipl_cadd_f

Computes the complex sum of two scalars.

Prototype

```
csipl_cscalar_f csipl_cadd_f(
                    csipl_cscalar_f x,
                    csipl_cscalar_f y);
```

Parameters

- x, complex scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := x + y.

Restrictions

Errors

Notes

csipl_rcadd_f

Computes the complex sum of two scalars.

Prototype

```
csipl_cscalar_f csipl_rcadd_f(
                    float           x,
                    csipl_cscalar_f y);
```

Parameters

- x, real scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := x + y.

Restrictions

Errors

Notes

csipl_cdiv_f

Computes the complex quotient of two scalars.

Prototype

```
csipl_cscalar_f csipl_cdiv_f(
                    csipl_cscalar_f x,
                    csipl_cscalar_f y);
```

Parameters

- x, complex scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := x/y.

Restrictions

The divisor must not be zero.

Errors

Notes

csipl_crdiv_f

Computes the complex quotient of two scalars.

Prototype

```
csipl_cscalar_f csipl_crdiv_f(
                    csipl_cscalar_f x,
                    float           y);
```

Parameters

- x, complex scalar, input.
- y, real scalar, input.

Return Value

- complex scalar.

Description

return value := x/y.

Restrictions

The divisor must not be zero.

Errors

Notes

csipl_cexp_f

Computes the exponential of a scalar.

Prototype

```
csipl_cscalar_f csipl_cexp_f(
                    csipl_cscalar_f A);
```

Parameters

- A, complex scalar, input.

Return Value

- complex scalar.

Description

return value := exp(A).

Restrictions

Overflow will occur if the argument is greater than the natural logarithm of the maximum representable number. Underflow will occur if the argument is less than the natural logarithm of the maximum representable number.

Errors

Notes

csipl_cjmul_f

Computes the product a complex scalar with the conjugate of a second complex scalar.

Prototype

```
csipl_cscalar_f csipl_cjmul_f(
                    csipl_cscalar_f x,
                    csipl_cscalar_f y);
```

Parameters

- x, complex scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := $x \cdot y^*$.

Restrictions

Errors

Notes

csipl_cmag_f

Computes the magnitude (absolute value) of a scalar.

Prototype

```
csipl_cscalar_f csipl_cmag_f(
                    csipl_cscalar_f A);
```

Parameters

- A, complex scalar, input.

Return Value

- complex scalar.

Description

return value := |A|.

Restrictions

Errors

Notes

csipl_cmagsq_f

Computes the magnitude squared of a complex scalar.

Prototype

```
float  csipl_cmagsq_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- real scalar.

Description

return value := $|x|^2$.

Restrictions

Errors

Notes

csipl_cmplx_f

Form a complex scalar from two real scalars.

Prototype

```
csipl_cscalar_f csipl_cmplx_f(
                    float  re,
                    float  im);
```

Parameters

- re, real scalar, input.
- im, real scalar, input.

Return Value

- complex scalar.

Description

return value := re + i · im.

Restrictions

Errors

Notes

csipl_cmul_f

Computes the complex product of two scalars.

Prototype

```
csipl_cscalar_f csipl_cmul_f(
                    csipl_cscalar_f x,
                    csipl_cscalar_f y);
```

Parameters

- x, complex scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := $x \cdot y$.

Restrictions

Errors

Notes

csipl_rcmul_f

Computes the complex product of two scalars.

Prototype

```
csipl_cscalar_f csipl_rcmul_f(
                    float           x,
                    csipl_cscalar_f y);
```

Parameters

- x, real scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := $x \cdot y$.

Restrictions

Errors

Notes

csipl_cneg_f

Computes the negation of a complex scalar.

Prototype

```
csipl_cscalar_f csipl_cneg_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- complex scalar.

Description

return value := −x.

Restrictions

Errors

Notes

csipl_conj_f

Computes the complex conjugate of a scalar.

Prototype

```
csipl_cscalar_f csipl_conj_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- complex scalar.

Description

return value := x^*.

Restrictions

Errors

Notes

csipl_crecip_f

Computes the reciprocal of a complex scalar.

Prototype

```
csipl_cscalar_f csipl_crecip_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- complex scalar.

Description

return value := 1/x.

Restrictions

The argument must not be zero.

Errors

Notes

csipl_csqrt_f

Computes the square root a complex scalar.

Prototype

```
csipl_cscalar_f csipl_csqrt_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- complex scalar.

Description

return value := $\sqrt{x}$.

Restrictions

Errors

Notes

csipl_csub_f

Computes the complex difference of two scalars.

Prototype

```
csipl_cscalar_f csipl_csub_f(
                    csipl_cscalar_f x,
                    csipl_cscalar_f y);
```

Parameters

- x, complex scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := x − y.

Restrictions

Errors

Notes

csipl_rcsub_f

Computes the complex difference of two scalars.

Prototype

```
csipl_cscalar_f csipl_rcsub_f(
                    float           x,
                    csipl_cscalar_f y);
```

Parameters

- x, real scalar, input.
- y, complex scalar, input.

Return Value

- complex scalar.

Description

return value := x − y.

Restrictions

Errors

Notes

csipl_crsub_f

Computes the complex difference of two scalars.

Prototype

```
csipl_cscalar_f csipl_crsub_f(
                    csipl_cscalar_f x,
                    float           y);
```

Parameters

- x, complex scalar, input.
- y, real scalar, input.

Return Value

- complex scalar.

Description

return value := x − y.

Restrictions

Errors

Notes

csipl_imag_f

Extract the imaginary part of a complex scalar.

Prototype

```
float  csipl_imag_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- real scalar.

Description

return value := imag(x).

Restrictions

Errors

Notes

csipl_polar_f

Convert a complex scalar from rectangular to polar form. The polar data consists of a real scalar containing the radius and a corresponding real scalar containing the argument (angle) of the complex scalar.

Prototype

```
void csipl_polar_f(
                    csipl_cscalar_f  a,
                    float           *r,
                    float           *t);
```

Parameters

- a, complex scalar, input.
- r, pointer to real scalar, output.
- t, pointer to real scalar, output.

Return Value

- none.

Description

r := |a| and t := arg(a).

Restrictions

The argument must be non-zero.

Errors

Notes

Complex numbers are always stored in rectangular $x + iy$ format. The polar form is represented by two real scalars.

csipl_real_f

Extract the real part of a complex scalar.

Prototype

```
float  csipl_real_f(
                    csipl_cscalar_f x);
```

Parameters

- x, complex scalar, input.

Return Value

- real scalar.

Description

return value := real(x).

Restrictions

Errors

Notes

csipl_rect_f

Convert a pair of real scalars from complex polar to complex rectangular form.

Prototype

```
csipl_cscalar_f csipl_rect_f(
                    float  r,
                    float  t);
```

Parameters

- r, real scalar, input.
- t, real scalar, input.

Return Value

- complex scalar.

Description

return value := $\mathtt{r} \cdot (\cos(\mathtt{t}) + \mathrm{i} \cdot \sin(\mathtt{t}))$.

Restrictions

Errors

Notes

Complex numbers are always stored in rectangular $x + \mathrm{i}y$ format. The polar form is represented by two real scalars.

4.3 Index Scalar Functions

- `csipl_matindex`
- `csipl_mcolindex`
- `csipl_mrowindex`

csipl_matindex

Form a matrix index from two vector indices.

Prototype

```
csipl_scalar_mi csipl_matindex(
                    csipl_index r,
                    csipl_index c);
```

Parameters

- r, vector-index scalar, input.
- c, vector-index scalar, input.

Return Value

- matrix-index scalar.

Description

return value := (r, c).

Restrictions

Errors

Notes

csipl_mcolindex

Returns the column vector index from a matrix index.

Prototype

```
csipl_index csipl_mcolindex(
                    csipl_scalar_mi mi);
```

Parameters

- `mi`, matrix-index scalar, input.

Return Value

- vector-index scalar.

Description

return value := column(`mi`).

Restrictions

Errors

Notes

csipl_mrowindex

Returns the row vector index from a matrix index.

Prototype

```
csipl_index csipl_mrowindex(
                    csipl_scalar_mi mi);
```

Parameters

- `mi`, matrix-index scalar, input.

Return Value

- vector-index scalar.

Description

return value := row(`mi`).

Restrictions

Errors

Notes

Chapter 5. Random Number Generation

5.1 Random Number Functions

- csipl_randcreate
- csipl_randdestroy
- csipl_randu_f
- csipl_crandu_f
- csipl_vrandu_f
- csipl_cvrandu_inter_f
- csipl_cvrandu_split_f
- csipl_randn_f
- csipl_crandn_f
- csipl_vrandn_f
- csipl_cvrandn_inter_f
- csipl_cvrandn_split_f

csipl_randcreate

Create a random number generator state object.

Prototype

```
csipl_randstate * csipl_randcreate(
                    csipl_index seed,
                    csipl_index numprocs,
                    csipl_index id,
                    csipl_rng   portable);
```

Parameters

- `seed`, vector-index scalar, input. Seed to initialise the generator.
- `numprocs`, vector-index scalar, input.
- `id`, vector-index scalar, input.
- `portable`, enumerated type, input.

`CSIPL_PRNG`	portable generator
`CSIPL_NPRNG`	non-portable generator

Return Value

- structure.

Description

Creates a state object for use by a random number generation function. The random number generator is characterised by specifying the number of random number generators (`numprocs`) the application is expected to create, and the index (`id`) of this generator. If the portable sequence is specified, then the number of random number generators specifies how many subsequences the primary sequence is partitioned into.

The function returns a random state object which holds the state information for the random number sequence generator, or `NULL` if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. $0 < \texttt{id} \leqslant \texttt{numprocs} \leqslant 2^{31} - 1$.

Notes

You must call this function for each random number sequence/stream the application needs. This might be one per processor, one per thread, etc. For the portable sequence to have the desired pseudo-random properties, each create must specify the same number of processors/subsequences.

csipl_randdestroy

Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure.

Prototype

```
int csipl_randdestroy(
                    csipl_randstate *rand);
```

Parameters

- rand, structure, input.

Return Value

- Error code.

Description

Destroys a random number state object.

Restrictions

Errors

The arguments must conform to the following:

1. The random number state object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

csipl_randu_f

Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.

Prototype

```
float  csipl_randu_f(
                    csipl_randstate *state);
```

Parameters

- `state`, structure, input.

Return Value

- real scalar.

Description

return value := uniform$(0, 1)$.

Restrictions

Errors

Notes

csipl_crandu_f

Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.

Prototype

```
csipl_cscalar_f csipl_crandu_f(
                    csipl_randstate *state);
```

Parameters

- `state`, structure, input.

Return Value

- complex scalar.

Description

return value := $\text{uniform}(0, 1) + \mathrm{i} \cdot \text{uniform}(0, 1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components where each component is $\text{uniform}(0, 1)$.

csipl_vrandu_f

Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.

Prototype

```
void csipl_vrandu_f(
                    csipl_randstate *state,
                    float           *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- `state`, structure, input.
- `R`, real vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`R`$[j * $`strideR`$] := \text{uniform}(0, 1)$.

Restrictions

Errors

Notes

csipl_cvrandu_inter_f

Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.

Prototype

```
void csipl_cvrandu_inter_f(
                    csipl_randstate *state,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- `state`, structure, input.
- `R`, complex vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`R`$[j * $`strideR`$] := \text{uniform}(0, 1) + \mathrm{i} \cdot \text{uniform}(0, 1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components where each component is $\text{uniform}(0, 1)$.

csipl_cvrandu_split_f

Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.

Prototype

```
void csipl_cvrandu_split_f(
                    csipl_randstate *state,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- `state`, structure, input.
- `R_re`, real part of complex vector, length n, output.
- `R_im`, imaginary part of complex vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`R`$[j * $`strideR`$] := \text{uniform}(0, 1) + \text{i} \cdot \text{uniform}(0, 1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components where each component is $\text{uniform}(0, 1)$.

csipl_randn_f

Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.

Prototype

```
float  csipl_randn_f(
                    csipl_randstate *state);
```

Parameters

- `state`, structure, input.

Return Value

- real scalar.

Description

return value := $\mathrm{N}(0, 1)$.

Restrictions

Errors

Notes

If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

csipl_crandn_f

Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.

Prototype

```
csipl_cscalar_f csipl_crandn_f(
                    csipl_randstate *state);
```

Parameters

- `state`, structure, input.

Return Value

- complex scalar.

Description

return value := $\mathrm{N}(0, 1) + \mathrm{i} \cdot \mathrm{N}(0, 1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components that are uncorrelated.

If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

csipl_vrandn_f

Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0,1)$. The random numbers are generated by summing values returned by the uniform random number generator.

Prototype

```
void csipl_vrandn_f(
                    csipl_randstate *state,
                    float           *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- `state`, structure, input.
- `R`, real vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`R`$[j * $`strideR`$] := \mathrm{N}(0,1)$.

Restrictions

Errors

Notes

If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

csipl_cvrandn_inter_f

Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0,1)$. The random numbers are generated by summing values returned by the uniform random number generator.

Prototype

```
void csipl_cvrandn_inter_f(
                    csipl_randstate *state,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- `state`, structure, input.
- `R`, complex vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`R`$[j * $`strideR`$] := \mathrm{N}(0,1) + \mathrm{i} \cdot \mathrm{N}(0,1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components that are uncorrelated.

If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

csipl_cvrandn_split_f

Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0,1)$. The random numbers are generated by summing values returned by the uniform random number generator.

Prototype

```
void csipl_cvrandn_split_f(
                    csipl_randstate *state,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- `state`, structure, input.
- `R_re`, real part of complex vector, length n, output.
- `R_im`, imaginary part of complex vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`R`$[j * $`strideR`$] := \mathrm{N}(0,1) + \mathrm{i} \cdot \mathrm{N}(0,1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components that are uncorrelated.

If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

Chapter 6. Vector And Elementwise Operations

6.1 Elementary Mathematical Functions

- csipl_vacos_f
- csipl_macos_f
- csipl_vasin_f
- csipl_masin_f
- csipl_vatan_f
- csipl_matan_f
- csipl_vatan2_f
- csipl_matan2_f
- csipl_vcos_f
- csipl_mcos_f
- csipl_vcosh_f
- csipl_mcosh_f
- csipl_vexp_f
- csipl_cvexp_inter_f
- csipl_cvexp_split_f
- csipl_mexp_f
- csipl_cmexp_inter_f
- csipl_cmexp_split_f
- csipl_vexp10_f
- csipl_mexp10_f
- csipl_vfloor_f
- csipl_vlog_f
- csipl_cvlog_inter_f
- csipl_cvlog_split_f
- csipl_mlog_f
- csipl_cmlog_inter_f
- csipl_cmlog_split_f
- csipl_vlog10_f

- csipl_mlog10_f
- csipl_vsin_f
- csipl_msin_f
- csipl_vsinh_f
- csipl_msinh_f
- csipl_vsqrt_f
- csipl_cvsqrt_inter_f
- csipl_cvsqrt_split_f
- csipl_msqrt_f
- csipl_cmsqrt_inter_f
- csipl_cmsqrt_split_f
- csipl_vtan_f
- csipl_mtan_f
- csipl_vtanh_f
- csipl_mtanh_f

csipl_vacos_f

Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a vector.

Prototype

```
void csipl_vacos_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \cos^{-1}(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

csipl_macos_f

Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a matrix.

Prototype

```
void csipl_macos_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \cos^{-1}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

csipl_vasin_f

Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a vector.

Prototype

```
void csipl_vasin_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \sin^{-1}(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

csipl_masin_f

Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a matrix.

Prototype

```
void csipl_masin_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sin^{-1}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

csipl_vatan_f

Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a vector.

Prototype

```
void csipl_vatan_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \tan^{-1}(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_matan_f

Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a matrix.

Prototype

```
void csipl_matan_f(
                    float        *A,
                    int           ldA,
                    float        *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan^{-1}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vatan2_f

Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input vectors.

Prototype

```
void csipl_vatan2_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B,
                    csipl_stride  strideB,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \tan^{-1}(\mathtt{A}[j * \mathtt{strideA}]/\mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

The rules for calculating the function value are the same as those for the ANSI C function `atan2`.

Restrictions

The arguments must not be both zero.

Errors

Notes

csipl_matan2_f

Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input matrices.

Prototype

```
void csipl_matan2_f(
                    float        *A,
                    int           ldA,
                    float        *B,
                    int           ldB,
                    float        *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan^{-1}(\mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

The rules for calculating the function value are the same as those for the ANSI C function `atan2`.

Restrictions

The arguments must not be both zero.

Errors

Notes

csipl_vcos_f

Computes the cosine for each element of a vector. Element angle values are in radians.

Prototype

```
void csipl_vcos_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \cos(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

csipl_mcos_f

Computes the cosine for each element of a matrix. Element angle values are in radians.

Prototype

```
void csipl_mcos_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \cos(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

csipl_vcosh_f

Computes the hyperbolic cosine for each element of a vector.

Prototype

```
void csipl_vcosh_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \cosh(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mcosh_f

Computes the hyperbolic cosine for each element of a matrix.

Prototype

```
void csipl_mcosh_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \cosh(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vexp_f

Computes the exponential function value for each element of a vector.

Prototype

```
void csipl_vexp_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \exp(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Overflow will occur if an element is greater than the natural logarithm of the largest representable number.

Underflow will occur if an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

csipl_cvexp_inter_f

Computes the exponential function value for each element of a vector.

Prototype

```
void csipl_cvexp_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \exp(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.

Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

csipl_cvexp_split_f

Computes the exponential function value for each element of a vector.

Prototype

```
void csipl_cvexp_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \exp(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.

Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

csipl_mexp_f

Computes the exponential function value for each element of a matrix.

Prototype

```
void csipl_mexp_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \exp(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow will occur if an element is greater than the natural logarithm of the largest representable number.

Underflow will occur if an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

csipl_cmexp_inter_f

Computes the exponential function value for each element of a matrix.

Prototype

```
void csipl_cmexp_inter_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \exp(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.

Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

csipl_cmexp_split_f

Computes the exponential function value for each element of a matrix.

Prototype

```
void csipl_cmexp_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \exp(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.

Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

csipl_vexp10_f

Computes the base 10 exponential for each element of a vector.

Prototype

```
void csipl_vexp10_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := 10^{\mathtt{A}[j*\mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

Overflow will occur if an element is greater than the base 10 logarithm of the largest representable number. Underflow will occur if an element is less than the negative of the base 10 logarithm of the largest representable number.

Errors

Notes

csipl_mexp10_f

Computes the base 10 exponential for each element of a matrix.

Prototype

```
void csipl_mexp10_f(
                    float        *A,
                    int           ldA,
                    float        *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := 10^{\mathtt{A}[j*\mathtt{ldA}+k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow will occur if an element is greater than the base 10 logarithm of the largest representable number. Underflow will occur if an element is less than the negative of the base 10 logarithm of the largest representable number.

Errors

Notes

csipl_vfloor_f

Computes the floor for each element of a vector.

Prototype

```
void csipl_vfloor_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \lfloor \mathtt{A}[j * \mathtt{strideA}] \rfloor$ where $0 \leqslant j < n$. Returns the largest integer less than or equal to the argument.

Restrictions

Errors

Notes

csipl_vlog_f

Computes the natural logarithm for each element of a vector.

Prototype

```
void csipl_vlog_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \log_e(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

csipl_cvlog_inter_f

Computes the natural logarithm for each element of a vector.

Prototype

```
void csipl_cvlog_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \log_{\mathrm{e}}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

For a complex value z where $0 \leqslant j < n$. we have $\log_{\mathrm{e}}(z) = \log_{\mathrm{e}}(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

csipl_cvlog_split_f

Computes the natural logarithm for each element of a vector.

Prototype

```
void csipl_cvlog_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \log_{\mathrm{e}}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

For a complex value z where $0 \leqslant j < n$. we have $\log_{\mathrm{e}}(z) = \log_{\mathrm{e}}(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

csipl_mlog_f

Computes the natural logarithm for each element of a matrix.

Prototype

```
void csipl_mlog_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_e(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

csipl_cmlog_inter_f

Computes the natural logarithm for each element of a matrix.

Prototype

```
void csipl_cmlog_inter_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_e(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

For a complex value z where $0 \leqslant j < m$ and $0 \leqslant k < n$. we have $\log_e(z) = \log_e(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < m$ and $0 \leqslant k < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

csipl_cmlog_split_f

Computes the natural logarithm for each element of a matrix.

Prototype

```
void csipl_cmlog_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_e(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

For a complex value z where $0 \leqslant j < m$ and $0 \leqslant k < n$. we have $\log_e(z) = \log_e(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < m$ and $0 \leqslant k < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

csipl_vlog10_f

Compute the base ten logarithm for each element of a vector.

Prototype

```
void csipl_vlog10_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \log_{10}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

The arguments must be greater than zero.

Errors

Notes

csipl_mlog10_f

Compute the base ten logarithm for each element of a matrix.

Prototype

```
void csipl_mlog10_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_{10}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must be greater than zero.

Errors

Notes

csipl_vsin_f

Compute the sine for each element of a vector. Element angle values are in radians.

Prototype

```
void csipl_vsin_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sin(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

csipl_msin_f

Compute the sine for each element of a matrix. Element angle values are in radians.

Prototype

```
void csipl_msin_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sin(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

csipl_vsinh_f

Computes the hyperbolic sine for each element of a vector.

Prototype

```
void csipl_vsinh_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sinh(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_msinh_f

Computes the hyperbolic sine for each element of a matrix.

Prototype

```
void csipl_msinh_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sinh(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vsqrt_f

Compute the square root for each element of a vector.

Prototype

```
void csipl_vsqrt_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sqrt{\mathtt{A}[j * \mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

The arguments must be greater than or equal to zero.

Errors

Notes

csipl_cvsqrt_inter_f

Compute the square root for each element of a vector.

Prototype

```
void csipl_cvsqrt_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sqrt{\mathtt{A}[j * \mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsqrt_split_f

Compute the square root for each element of a vector.

Prototype

```
void csipl_cvsqrt_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sqrt{\mathtt{A}[j * \mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_msqrt_f

Compute the square root for each element of a matrix.

Prototype

```
void csipl_msqrt_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sqrt{\mathtt{A}[j * \mathtt{ldA} + k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must be greater than or equal to zero.

Errors

Notes

csipl_cmsqrt_inter_f

Compute the square root for each element of a matrix.

Prototype

```
void csipl_cmsqrt_inter_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sqrt{\mathtt{A}[j * \mathtt{ldA} + k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmsqrt_split_f

Compute the square root for each element of a matrix.

Prototype

```
void csipl_cmsqrt_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sqrt{\mathtt{A}[j * \mathtt{ldA} + k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vtan_f

Compute the tangent for each element of a vector. Element angle values are in radians.

Prototype

```
void csipl_vtan_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R[$j *$ strideR] := tan(A[$j *$ strideA]) where $0 \leqslant j < n$.

Restrictions

For element values $(n + 1/2)\pi$, the tangent function has a singularity and is undefined.

Errors

Notes

csipl_mtan_f

Compute the tangent for each element of a matrix. Element angle values are in radians.

Prototype

```
void csipl_mtan_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

For element values $(n + 1/2)\pi$, the tangent function has a singularity and is undefined.

Errors

Notes

csipl_vtanh_f

Computes the hyperbolic tangent for each element of a vector.

Prototype

```
void csipl_vtanh_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \tanh(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mtanh_f

Computes the hyperbolic tangent for each element of a matrix.

Prototype

```
void csipl_mtanh_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tanh(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

6.2 Unary Operations

- csipl_varg_f
- csipl_marg_f
- csipl_vceil_f
- csipl_cvconj_inter_f
- csipl_cvconj_split_f
- csipl_cmconj_inter_f
- csipl_cmconj_split_f
- csipl_vcumsum_P
- csipl_cvcumsum_inter_P
- csipl_cvcumsum_split_P
- csipl_mcumsum_P
- csipl_cmcumsum_inter_P
- csipl_cmcumsum_split_P
- csipl_veuler_f
- csipl_meuler_f
- csipl_vmag_P
- csipl_cvmag_inter_f
- csipl_cvmag_split_f
- csipl_mmag_f
- csipl_cmmag_inter_f
- csipl_cmmag_split_f
- csipl_vcmagsq_inter_f
- csipl_vcmagsq_split_f
- csipl_mcmagsq_f
- csipl_vmeanval_f
- csipl_cvmeanval_inter_f
- csipl_cvmeanval_split_f
- csipl_mmeanval_f
- csipl_cmmeanval_inter_f
- csipl_cmmeanval_split_f
- csipl_vmeansqval_f
- csipl_cvmeansqval_inter_f
- csipl_cvmeansqval_split_f

- csipl_mmeansqval_f
- csipl_cmmeansqval_inter_f
- csipl_cmmeansqval_split_f
- csipl_vmodulate_f
- csipl_cvmodulate_inter_f
- csipl_cvmodulate_split_f
- csipl_vneg_P
- csipl_cvneg_inter_f
- csipl_cvneg_split_f
- csipl_mneg_P
- csipl_cmneg_inter_f
- csipl_cmneg_split_f
- csipl_vrecip_f
- csipl_cvrecip_inter_f
- csipl_cvrecip_split_f
- csipl_mrecip_f
- csipl_cmrecip_inter_f
- csipl_cmrecip_split_f
- csipl_vrsqrt_f
- csipl_mrsqrt_f
- csipl_vsq_f
- csipl_msq_f
- csipl_vsumval_P
- csipl_cvsumval_inter_P
- csipl_cvsumval_split_P
- csipl_msumval_P
- csipl_cmsumval_inter_P
- csipl_cmsumval_split_P
- csipl_vsumval_bl
- csipl_vsumsqval_f
- csipl_msumsqval_f

csipl_varg_f

Computes the argument in radians $[-\pi, \pi]$ for each element of a complex vector.

Prototype

```
void csipl_varg_f(
                    void          *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \tan^{-1}(\mathrm{imag}(\mathtt{A}[j * \mathtt{strideA}])/\mathrm{real}(\mathtt{A}[j * \mathtt{strideA}]))$.

Restrictions

Errors

Notes

This function is based on csipl_Satan2_f.

csipl_marg_f

Computes the argument in radians $[-\pi, \pi]$ for each element of a complex matrix.

Prototype

```
void csipl_marg_f(
                    void          *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan^{-1}(\mathrm{imag}(\mathtt{A}[j * \mathtt{ldA} + k])/\mathrm{real}(\mathtt{A}[j * \mathtt{ldA} + k]))$.

Restrictions

Errors

Notes

This function is based on csipl_Satan2_f.

csipl_vceil_f

Computes the ceiling for each element of a vector.

Prototype

```
void csipl_vceil_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \lceil \mathtt{A}[j * \mathtt{strideA}] \rceil$ where $0 \leqslant j < n$. Returns the smallest integer greater than or equal to the argument.

Restrictions

Errors

Notes

csipl_cvconj_inter_f

Compute the conjugate for each element of a complex vector.

Prototype

```
void csipl_cvconj_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvconj_split_f

Compute the conjugate for each element of a complex vector.

Prototype

```
void csipl_cvconj_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cmconj_inter_f

Compute the conjugate for each element of a complex matrix.

Prototype

```
void csipl_cmconj_inter_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmconj_split_f

Compute the conjugate for each element of a complex matrix.

Prototype

```
void csipl_cmconj_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vcumsum_P

Compute the cumulative sum of the elements of a vector.

Prototype

```
void csipl_vcumsum_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

csipl_vcumsum_f

csipl_vcumsum_i

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j] := \sum_{i=0}^{j} \mathtt{A}[i]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_cvcumsum_inter_P

Compute the cumulative sum of the elements of a vector.

Prototype

```
void csipl_cvcumsum_inter_P(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cvcumsum_inter_f
csipl_cvcumsum_inter_i
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j] := \sum_{i=0}^{j} \mathtt{A}[i]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_cvcumsum_split_P

Compute the cumulative sum of the elements of a vector.

Prototype

```
void csipl_cvcumsum_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_cvcumsum_split_f
- csipl_cvcumsum_split_i

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j] := \sum_{i=0}^{j} \mathtt{A}[i]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_mcumsum_P

Compute the cumulative sums of the elements in the rows or columns of a matrix.

Prototype

```
void csipl_mcumsum_P(
                     csipl_major    dir,
                     scalar_P      *R,
                     int            ldR,
                     csipl_length   m,
                     csipl_length   n);
```

The following instances are supported:

csipl_mcumsum_f

csipl_mcumsum_i

Parameters

- dir, enumerated type, input.

  CSIPL_ROW apply operation to the rows
  CSIPL_COL apply operation to the columns
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Row:
$\mathtt{R}[j,k] := \sum_{i=0}^{k} \mathtt{A}[j,i]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Column:
$\mathtt{R}[j,k] := \sum_{i=0}^{j} \mathtt{A}[i,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_cmcumsum_inter_P

Compute the cumulative sum of the elements of a vector.

Prototype

```
void csipl_cmcumsum_inter_P(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cmcumsum_inter_f
csipl_cmcumsum_inter_i
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j] := \sum_{i=0}^{j} \mathtt{A}[i]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_cmcumsum_split_P

Compute the cumulative sum of the elements of a vector.

Prototype

```
void csipl_cmcumsum_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_cmcumsum_split_f
- csipl_cmcumsum_split_i

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j] := \sum_{i=0}^{j} \mathtt{A}[i]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_veuler_f

Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a vector.

Prototype

```
void csipl_veuler_f(
                    float        *A,
                    csipl_stride  strideA,
                    void         *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \cos(\mathtt{A}[j * \mathtt{strideA}]) + \mathrm{i} \cdot \sin(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The speed may be adversely affected for large arguments because of conversion of the argument to its principal value.

csipl_meuler_f

Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a matrix.

Prototype

```
void csipl_meuler_f(
                    float         *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j*\mathtt{ldR}+k] := \cos(\mathtt{A}[j*\mathtt{ldA}+k])+\mathrm{i}\cdot\sin(\mathtt{A}[j*\mathtt{ldA}+k]))$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

The speed may be adversely affected for large arguments because of conversion of the argument to its principal value.

csipl_vmag_P

Compute the magnitude for each element of a vector.

Prototype

```
void csipl_vmag_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vmag_f
- csipl_vmag_i
- csipl_vmag_si

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmag_inter_f

Compute the magnitude for each element of a vector.

Prototype

```
void csipl_cvmag_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmag_split_f

Compute the magnitude for each element of a vector.

Prototype

```
void csipl_cvmag_split_f(
                    float         *A_re,
                    float         *A_im,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mmag_f

Compute the magnitude for each element of a matrix.

Prototype

```
void csipl_mmag_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmmag_inter_f

Compute the magnitude for each element of a matrix.

Prototype

```
void csipl_cmmag_inter_f(
                    void          *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmmag_split_f

Compute the magnitude for each element of a matrix.

Prototype

```
void csipl_cmmag_split_f(
                    float         *A_re,
                    float         *A_im,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vcmagsq_inter_f

Computes the square of the magnitudes for each element of a vector.

Prototype

```
void csipl_vcmagsq_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|^2$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vcmagsq_split_f

Computes the square of the magnitudes for each element of a vector.

Prototype

```
void csipl_vcmagsq_split_f(
                    float         *A_re,
                    float         *A_im,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|^2$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mcmagsq_f

Computes the square of the magnitudes for each element of a matrix.

Prototype

```
void csipl_mcmagsq_f(
                     void          *A,
                     int            ldA,
                     float         *R,
                     int            ldR,
                     csipl_length   m,
                     csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vmeanval_f

Returns the mean value of the elements of a vector.

Prototype

```
float  csipl_vmeanval_f(
                    float         *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum$ A$[j *$ strideA$]$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cvmeanval_inter_f

Returns the mean value of the elements of a vector.

Prototype

```
csipl_cscalar_f csipl_cvmeanval_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value $:= 1/N \sum$ A$[j *$ strideA$]$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cvmeanval_split_f

Returns the mean value of the elements of a vector.

Prototype

```
csipl_cscalar_f csipl_cvmeanval_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value $:= 1/N \sum$ A$[j *$ strideA$]$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_mmeanval_f

Returns the mean value of the elements of a matrix.

Prototype

```
float  csipl_mmeanval_f(
                    float        *A,
                    int           ldA,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cmmeanval_inter_f

Returns the mean value of the elements of a matrix.

Prototype

```
csipl_cscalar_f csipl_cmmeanval_inter_f(
                    void          *A,
                    int            ldA,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value $:= 1/N \sum \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cmmeanval_split_f

Returns the mean value of the elements of a matrix.

Prototype

```
csipl_cscalar_f csipl_cmmeanval_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value $:= 1/N \sum \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_vmeansqval_f

Returns the mean magnitude squared value of the elements of a vector.

Prototype

```
float  csipl_vmeansqval_f(
                    float        *A,
                    csipl_stride  strideA,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{strideA}]|^2$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cvmeansqval_inter_f

Returns the mean magnitude squared value of the elements of a vector.

Prototype

```
float  csipl_cvmeansqval_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{strideA}]|^2$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cvmeansqval_split_f

Returns the mean magnitude squared value of the elements of a vector.

Prototype

```
float  csipl_cvmeansqval_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{strideA}]|^2$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_mmeansqval_f

Returns the mean magnitude squared value of the elements of a matrix.

Prototype

```
float  csipl_mmeansqval_f(
                    float        *A,
                    int           ldA,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{ldA} + k]|^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cmmeansqval_inter_f

Returns the mean magnitude squared value of the elements of a matrix.

Prototype

```
float  csipl_cmmeansqval_inter_f(
                    void          *A,
                    int            ldA,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{ldA} + k]|^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_cmmeansqval_split_f

Returns the mean magnitude squared value of the elements of a matrix.

Prototype

```
float  csipl_cmmeansqval_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{ldA} + k]|^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

csipl_vmodulate_f

Computes the modulation of a real vector by a specified complex frequency.

Prototype

```
float  csipl_vmodulate_f(
                    float         *A,
                    csipl_stride   strideA,
                    float          nu,
                    float          phi,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- nu, real scalar, input.
- phi, real scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

$\mathtt{R}[j] := \mathtt{A}[j] \cdot (\cos(i \cdot \mathtt{nu} + \mathtt{phi}) + \mathrm{i} \cdot \sin(i \cdot \mathtt{nu} + \mathtt{phi}))$ where $0 \leqslant j < n$,

nu is the frequency in radians per index, and phi is the initial phase.

The function returns $N \cdot \mathtt{nu} + \mathtt{phi}$ where N is the length of the vector. This can be used as the initial phase in the nect call to provide a continuous modulation.

Restrictions

Errors

Notes

To provide continuous filtering but processed by frames the return value can be used as the initial phase for the next frame.

csipl_cvmodulate_inter_f

Computes the modulation of a real vector by a specified complex frequency.

Prototype

```
float  csipl_cvmodulate_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    float          nu,
                    float          phi,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- nu, real scalar, input.
- phi, real scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

$\mathtt{R}[j] := \mathtt{A}[j] \cdot (\cos(i \cdot \mathtt{nu} + \mathtt{phi}) + \mathrm{i} \cdot \sin(i \cdot \mathtt{nu} + \mathtt{phi}))$ where $0 \leqslant j < n$,

nu is the frequency in radians per index, and phi is the initial phase.

The function returns $N \cdot \mathtt{nu} + \mathtt{phi}$ where N is the length of the vector. This can be used as the initial phase in the nect call to provide a continuous modulation.

Restrictions

Errors

Notes

To provide continuous filtering but processed by frames the return value can be used as the initial phase for the next frame.

csipl_cvmodulate_split_f

Computes the modulation of a real vector by a specified complex frequency.

Prototype

```
float  csipl_cvmodulate_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float         nu,
                    float         phi,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- nu, real scalar, input.
- phi, real scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

$\mathtt{R}[j] := \mathtt{A}[j] \cdot (\cos(i \cdot \mathtt{nu} + \mathtt{phi}) + \mathrm{i} \cdot \sin(i \cdot \mathtt{nu} + \mathtt{phi}))$ where $0 \leqslant j < n$,

nu is the frequency in radians per index, and phi is the initial phase.

The function returns $N \cdot \mathtt{nu} + \mathtt{phi}$ where N is the length of the vector. This can be used as the initial phase in the nect call to provide a continuous modulation.

Restrictions

Errors

Notes

To provide continuous filtering but processed by frames the return value can be used as the initial phase for the next frame.

csipl_vneg_P

Computes the negation for each element of a vector.

Prototype

```
void csipl_vneg_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vneg_f
- csipl_vneg_i
- csipl_vneg_si

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := -\mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvneg_inter_f

Computes the negation for each element of a vector.

Prototype

```
void csipl_cvneg_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := -\mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvneg_split_f

Computes the negation for each element of a vector.

Prototype

```
void csipl_cvneg_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := -\mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mneg_P

Computes the negation for each element of a matrix.

Prototype

```
void csipl_mneg_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_mneg_f
csipl_mneg_i
```

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := -\mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmneg_inter_f

Computes the negation for each element of a matrix.

Prototype

```
void csipl_cmneg_inter_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := -\mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmneg_split_f

Computes the negation for each element of a matrix.

Prototype

```
void csipl_cmneg_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := -\mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vrecip_f

Computes the reciprocal for each element of a vector.

Prototype

```
void csipl_vrecip_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := 1/\mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

csipl_cvrecip_inter_f

Computes the reciprocal for each element of a vector.

Prototype

```
void csipl_cvrecip_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := 1/\mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

csipl_cvrecip_split_f

Computes the reciprocal for each element of a vector.

Prototype

```
void csipl_cvrecip_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := 1/\texttt{A}[j * \texttt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

csipl_mrecip_f

Computes the reciprocal for each element of a matrix.

Prototype

```
void csipl_mrecip_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := 1/\mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

csipl_cmrecip_inter_f

Computes the reciprocal for each element of a matrix.

Prototype

```
void csipl_cmrecip_inter_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := 1/\mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

csipl_cmrecip_split_f

Computes the reciprocal for each element of a matrix.

Prototype

```
void csipl_cmrecip_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := 1/\mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

csipl_vrsqrt_f

Computes the reciprocal of the square root for each element of a vector.

Prototype

```
void csipl_vrsqrt_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := 1/\sqrt{\mathtt{A}[j * \mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

csipl_mrsqrt_f

Computes the reciprocal of the square root for each element of a matrix.

Prototype

```
void csipl_mrsqrt_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := 1/\sqrt{\mathtt{A}[j * \mathtt{ldA} + k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

csipl_vsq_f

Computes the square for each element of a vector.

Prototype

```
void csipl_vsq_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]^2$ where $0 \leqslant j < n$.

Restrictions

Overflow will occur if an element's magnitude is greater than the square root of the largest representable number. Underflow will occur if an element's magnitude is less than the square root of the minimum smallest representable number.

Errors

Notes

csipl_msq_f

Computes the square for each element of a matrix.

Prototype

```
void csipl_msq_f(
                    float         *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow will occur if an element's magnitude is greater than the square root of the largest representable number. Underflow will occur if an element's magnitude is less than the square root of the minimum smallest representable number.

Errors

Notes

csipl_vsumval_P

Returns the sum of the elements of a vector.

Prototype

```
scalar_P  csipl_vsumval_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vsumval_f
- csipl_vsumval_i
- csipl_vsumval_si

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_cvsumval_inter_P

Returns the sum of the elements of a vector.

Prototype

```
scalar_P  csipl_cvsumval_inter_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

The following instances are supported:

csipl_cvsumval_inter_f

csipl_cvsumval_inter_i

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_cvsumval_split_P

Returns the sum of the elements of a vector.

Prototype

```
scalar_P  csipl_cvsumval_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    csipl_length   n);
```

The following instances are supported:

csipl_cvsumval_split_f

csipl_cvsumval_split_i

Parameters

- A_re, real part of vector, length n, input.
- A_im, imaginary part of vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_msumval_P

Returns the sum of the elements of a vector.

Prototype

```
scalar_P  csipl_msumval_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_msumval_f
csipl_msumval_i
```

Parameters

- A, matrix, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_cmsumval_inter_P

Returns the sum of the elements of a vector.

Prototype

```
scalar_P  csipl_cmsumval_inter_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cmsumval_inter_f
csipl_cmsumval_inter_i
```

Parameters

- A, matrix, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_cmsumval_split_P

Returns the sum of the elements of a vector.

Prototype

```
scalar_P  csipl_cmsumval_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cmsumval_split_f
csipl_cmsumval_split_i
```

Parameters

- A_re, real part of matrix, length n, input.
- A_im, imaginary part of matrix, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_vsumval_bl

Returns the sum of the elements of a vector.

Prototype

```
csipl_scalar_bl csipl_vsumval_bl(
                    signed int    *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, boolean vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- boolean scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

A boolean value is defined so that zero is false and anything else is true. This function returns the number of non-false values in a boolean object. The return type is an unsigned integer type large enough to represent the size of a vector.

csipl_vsumsqval_f

Returns the sum of the squares of the elements of a vector.

Prototype

```
float  csipl_vsumsqval_f(
                    float        *A,
                    csipl_stride  strideA,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]^2$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_msumsqval_f

Returns the sum of the squares of the elements of a matrix.

Prototype

```
float  csipl_msumsqval_f(
                    float         *A,
                    int            ldA,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{ldA} + k]^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

6.3 Binary Operations

- csipl_vadd_P
- csipl_cvadd_inter_P
- csipl_cvadd_split_P
- csipl_madd_P
- csipl_cmadd_inter_f
- csipl_cmadd_split_f
- csipl_rcvadd_inter_f
- csipl_rcvadd_split_f
- csipl_rcmadd_inter_f
- csipl_rcmadd_split_f
- csipl_svadd_P
- csipl_csvadd_inter_f
- csipl_csvadd_split_f
- csipl_smadd_P
- csipl_csmadd_inter_f
- csipl_csmadd_split_f
- csipl_rscvadd_inter_f
- csipl_rscvadd_split_f
- csipl_rscmadd_inter_f
- csipl_rscmadd_split_f
- csipl_Dvdiv_P
- csipl_Dvdiv_inter_P
- csipl_Dvdiv_split_P
- csipl_mdiv_f
- csipl_cmdiv_inter_f
- csipl_cmdiv_split_f
- csipl_rcvdiv_inter_f
- csipl_rcvdiv_split_f
- csipl_rcmdiv_inter_f
- csipl_rcmdiv_split_f
- csipl_crmdiv_inter_f
- csipl_crmdiv_split_f
- csipl_rscmsub_inter_f

- csipl_rscmsub_split_f
- csipl_vsdiv_f
- csipl_cvrsdiv_inter_f
- csipl_cvrsdiv_split_f
- csipl_msdiv_f
- csipl_cmrsdiv_inter_f
- csipl_cmrsdiv_split_f
- csipl_vexpoavg_f
- csipl_cvexpoavg_inter_f
- csipl_cvexpoavg_split_f
- csipl_mexpoavg_f
- csipl_cmexpoavg_inter_f
- csipl_cmexpoavg_split_f
- csipl_vhypot_f
- csipl_mhypot_f
- csipl_cvjmul_inter_f
- csipl_cvjmul_split_f
- csipl_cmjmul_inter_f
- csipl_cmjmul_split_f
- csipl_Dvmul_P
- csipl_cvmul_inter_f
- csipl_cvmul_split_f
- csipl_mmul_P
- csipl_cmmul_inter_f
- csipl_cmmul_split_f
- csipl_rcvmul_inter_f
- csipl_rcvmul_split_f
- csipl_rcmmul_inter_f
- csipl_rcmmul_split_f
- csipl_rscvmul_inter_f
- csipl_rscvmul_split_f
- csipl_csvmul_inter_f
- csipl_csvmul_split_f
- csipl_smmul_f

- csipl_csmmul_inter_f
- csipl_csmmul_split_f
- csipl_rscmmul_inter_f
- csipl_rscmmul_split_f
- csipl_vmmul_f
- csipl_cvmmul_inter_f
- csipl_cvmmul_split_f
- csipl_rvcmmul_inter_f
- csipl_rvcmmul_split_f
- csipl_vsub_P
- csipl_cvsub_inter_f
- csipl_cvsub_split_f
- csipl_msub_P
- csipl_cmsub_inter_f
- csipl_cmsub_split_f
- csipl_crmsub_inter_f
- csipl_crmsub_split_f
- csipl_rcvsub_inter_f
- csipl_rcvsub_split_f
- csipl_rcmsub_inter_f
- csipl_rcmsub_split_f
- csipl_crvsub_inter_f
- csipl_crvsub_split_f
- csipl_svsub_P
- csipl_csvsub_inter_f
- csipl_csvsub_split_f
- csipl_smsub_P
- csipl_csmsub_inter_f
- csipl_csmsub_split_f
- csipl_smdiv_f
- csipl_csmdiv_inter_f
- csipl_csmdiv_split_f
- csipl_rscvdiv_inter_f
- csipl_rscvdiv_split_f

- csipl_csvdiv_inter_f
- csipl_csvdiv_split_f
- csipl_rscvsub_inter_f
- csipl_rscvsub_split_f
- csipl_rscmdiv_inter_f
- csipl_rscmdiv_split_f

csipl_vadd_P

Computes the sum, by element, of two vectors.

Prototype

```
void csipl_vadd_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vadd_f
- csipl_vadd_i
- csipl_vadd_si

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvadd_inter_P

Computes the sum, by element, of two vectors.

Prototype

```
void csipl_cvadd_inter_P(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

`csipl_cvadd_inter_f`

`csipl_cvadd_inter_i`

Parameters

- `A`, complex vector, length n, input.
- `strideA`, integer scalar, input.
- `B`, complex vector, length n, input.
- `strideB`, integer scalar, input.
- `R`, complex vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvadd_split_P

Computes the sum, by element, of two vectors.

Prototype

```
void csipl_cvadd_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    csipl_stride   strideB,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

`csipl_cvadd_split_f`

`csipl_cvadd_split_i`

Parameters

- `A_re`, real part of complex vector, length n, input.
- `A_im`, imaginary part of complex vector, length n, input.
- `strideA`, integer scalar, input.
- `B_re`, real part of complex vector, length n, input.
- `B_im`, imaginary part of complex vector, length n, input.
- `strideB`, integer scalar, input.
- `R_re`, real part of complex vector, length n, output.
- `R_im`, imaginary part of complex vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}] + \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_madd_P

Computes the sum, by element, of two matrices.

Prototype

```
void csipl_madd_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

csipl_madd_f

csipl_madd_i

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmadd_inter_f

Computes the sum, by element, of two matrices.

Prototype

```
void csipl_cmadd_inter_f(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmadd_split_f

Computes the sum, by element, of two matrices.

Prototype

```
void csipl_cmadd_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rcvadd_inter_f

Computes the sum, by element, of two vectors.

Prototype

```
void csipl_rcvadd_inter_f(
                    float        *A,
                    csipl_stride  strideA,
                    void         *B,
                    csipl_stride  strideB,
                    void         *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rcvadd_split_f

Computes the sum, by element, of two vectors.

Prototype

```
void csipl_rcvadd_split_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B_re,
                    float         *B_im,
                    csipl_stride   strideB,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rcmadd_inter_f

Computes the sum, by element, of two matrices.

Prototype

```
void csipl_rcmadd_inter_f(
                    float        *A,
                    int           ldA,
                    void         *B,
                    int           ldB,
                    void         *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rcmadd_split_f

Computes the sum, by element, of two matrices.

Prototype

```
void csipl_rcmadd_split_f(
                    float         *A,
                    int            ldA,
                    float         *B_re,
                    float         *B_im,
                    int            ldB,
                    float         *R_re,
                    float         *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_svadd_P

Computes the sum, by element, of a scalar and a vector.

Prototype

```
void csipl_svadd_P(
                    scalar_P       a,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_svadd_f
- csipl_svadd_i
- csipl_svadd_si

Parameters

- a, scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_csvadd_inter_f

Computes the sum, by element, of a scalar and a vector.

Prototype

```
void csipl_csvadd_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    csipl_stride     strideB,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \texttt{a} + \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_csvadd_split_f

Computes the sum, by element, of a scalar and a vector.

Prototype

```
void csipl_csvadd_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    csipl_stride     strideB,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_smadd_P

Computes the sum, by element, of a scalar and a matrix.

Prototype

```
void csipl_smadd_P(
                    scalar_P       a,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_smadd_f
csipl_smadd_i
```

Parameters

- a, scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_csmadd_inter_f

Computes the sum, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmadd_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    int              ldB,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_csmadd_split_f

Computes the sum, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmadd_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    int              ldB,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rscvadd_inter_f

Computes the sum, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvadd_inter_f(
                    float          a,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rscvadd_split_f

Computes the sum, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvadd_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    csipl_stride   strideB,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rscmadd_inter_f

Computes the sum, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmadd_inter_f(
                    float          a,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rscmadd_split_f

Computes the sum, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmadd_split_f(
                    float         a,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_Dvdiv_P

Computes the quotient, by element, of two vectors.

Prototype

```
void csipl_Dvdiv_P(
                    csipl_Dvview_P *A,
                    csipl_stride    strideA,
                    csipl_Dvview_P *B,
                    csipl_stride    strideB,
                    csipl_Dvview_P *R,
                    csipl_stride    strideR,
                    csipl_length    n);
```

The following instances are supported:

csipl_vdiv_f

csipl_svdiv_f

Parameters

- A, real or complex vector, length n, input.
- strideA, integer scalar, input.
- B, real or complex vector, length n, input.
- strideB, integer scalar, input.
- R, real or complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_Dvdiv_inter_P

Computes the quotient, by element, of two vectors.

Prototype

```
void csipl_Dvdiv_inter_P(
                    csipl_Dvview_P *A,
                    csipl_stride    strideA,
                    csipl_Dvview_P *B,
                    csipl_stride    strideB,
                    csipl_Dvview_P *R,
                    csipl_stride    strideR,
                    csipl_length    n);
```

The following instances are supported:

`csipl_crvdiv_inter_f`

`csipl_cvdiv_inter_f`

Parameters

- `A`, real or complex vector, length n, input.
- `strideA`, integer scalar, input.
- `B`, real or complex vector, length n, input.
- `strideB`, integer scalar, input.
- `R`, real or complex vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_Dvdiv_split_P

Computes the quotient, by element, of two vectors.

Prototype

```
void csipl_Dvdiv_split_P(
                    csipl_Dvview_P *A_re,
                    csipl_Dvview_P *A_im,
                    csipl_stride    strideA,
                    csipl_Dvview_P *B_re,
                    csipl_Dvview_P *B_im,
                    csipl_stride    strideB,
                    csipl_Dvview_P *R_re,
                    csipl_Dvview_P *R_im,
                    csipl_stride    strideR,
                    csipl_length    n);
```

The following instances are supported:

```
csipl_crvdiv_split_f
csipl_cvdiv_split_f
```

Parameters

- A_re, real part of real or complex vector, length n, input.
- A_im, imaginary part of real or complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of real or complex vector, length n, input.
- B_im, imaginary part of real or complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of real or complex vector, length n, output.
- R_im, imaginary part of real or complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}]/\texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_mdiv_f

Computes the quotient, by element, of two matrices.

Prototype

```
void csipl_mdiv_f(
                    float         *A,
                    int            ldA,
                    float         *B,
                    int            ldB,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_cmdiv_inter_f

Computes the quotient, by element, of two matrices.

Prototype

```
void csipl_cmdiv_inter_f(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_cmdiv_split_f

Computes the quotient, by element, of two matrices.

Prototype

```
void csipl_cmdiv_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_rcvdiv_inter_f

Computes the quotient, by element, of two vectors.

Prototype

```
void csipl_rcvdiv_inter_f(
                    float          a,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a}/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_rcvdiv_split_f

Computes the quotient, by element, of two vectors.

Prototype

```
void csipl_rcvdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    csipl_stride   strideB,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a}/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_rcmdiv_inter_f

Computes the quotient, by element, of two matrices.

Prototype

```
void csipl_rcmdiv_inter_f(
                    float          a,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_rcmdiv_split_f

Computes the quotient, by element, of two matrices.

Prototype

```
void csipl_rcmdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    int            ldB,
                    float         *R_re,
                    float         *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_crmdiv_inter_f

Computes the quotient, by element, of two matrices.

Prototype

```
void csipl_crmdiv_inter_f(
                    void          *A,
                    int            ldA,
                    float         *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_crmdiv_split_f

Computes the quotient, by element, of two matrices.

Prototype

```
void csipl_crmdiv_split_f(
                    float         *A_re,
                    float         *A_im,
                    int            ldA,
                    float         *B,
                    int            ldB,
                    float         *R_re,
                    float         *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_rscmsub_inter_f

Computes the difference, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmsub_inter_f(
                    float          a,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rscmsub_split_f

Computes the difference, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmsub_split_f(
                    float         a,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vsdiv_f

Computes the quotient, by element, of a vector and a scalar.

Prototype

```
void csipl_vsdiv_f(
                    float         *A,
                    csipl_stride   strideA,
                    float          b,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- b, real scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]/\mathtt{b}$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_cvrsdiv_inter_f

Computes the quotient, by element, of a vector and a scalar.

Prototype

```
void csipl_cvrsdiv_inter_f(
                    float         *A,
                    csipl_stride   strideA,
                    float          b,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- b, real scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]/\mathtt{b}$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_cvrsdiv_split_f

Computes the quotient, by element, of a vector and a scalar.

Prototype

```
void csipl_cvrsdiv_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float         b_re,
                    float         b_im,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of real vector, length n, input.
- A_im, imaginary part of real vector, length n, input.
- strideA, integer scalar, input.
- b_re, real part of real scalar, input.
- b_im, imaginary part of real scalar, input.
- R_re, real part of real vector, length n, output.
- R_im, imaginary part of real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] :=$ A$[j * \texttt{strideA}]/$b where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_msdiv_f

Computes the quotient, by element, of a matrix and a scalar.

Prototype

```
void csipl_msdiv_f(
                    float         *A,
                    int            ldA,
                    float          b,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- b, real scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{b}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_cmrsdiv_inter_f

Computes the quotient, by element, of a matrix and a scalar.

Prototype

```
void csipl_cmrsdiv_inter_f(
                    void            *A,
                    int              ldA,
                    csipl_cscalar_f  b,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- b, complex scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{b}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_cmrsdiv_split_f

Computes the quotient, by element, of a matrix and a scalar.

Prototype

```
void csipl_cmrsdiv_split_f(
                    float           *A_re,
                    float           *A_im,
                    int              ldA,
                    csipl_cscalar_f  b_re,
                    csipl_cscalar_f  b_im,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- b_re, real part of complex scalar, input.
- b_im, imaginary part of complex scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{b}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_vexpoavg_f

Computes an exponential weighted average, by element, of two vectors.

Prototype

```
void csipl_vexpoavg_f(
                    float          a,
                    float         *B,
                    csipl_stride   strideB,
                    float         *C,
                    csipl_stride   strideC,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- C, real vector, length n, modified in place.
- strideC, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{C}[j * \mathtt{strideC}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}] + (1 - \mathtt{a}) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvexpoavg_inter_f

Computes an exponential weighted average, by element, of two vectors.

Prototype

```
void csipl_cvexpoavg_inter_f(
                    float          a,
                    void          *B,
                    csipl_stride   strideB,
                    void          *C,
                    csipl_stride   strideC,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- C, complex vector, length n, modified in place.
- strideC, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{C}[j * \mathtt{strideC}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}] + (1 - \mathtt{a}) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvexpoavg_split_f

Computes an exponential weighted average, by element, of two vectors.

Prototype

```
void csipl_cvexpoavg_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    csipl_stride   strideB,
                    float         *C_re,
                    float         *C_im,
                    csipl_stride   strideC,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- C_re, real part of complex vector, length n, modified in place.
- C_im, imaginary part of complex vector, length n, modified in place.
- strideC, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{C}[j * \mathtt{strideC}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}] + (1 - \mathtt{a}) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mexpoavg_f

Computes an exponential weighted average, by element, of two matrices.

Prototype

```
void csipl_mexpoavg_f(
                    float          a,
                    float         *B,
                    int            ldB,
                    float         *C,
                    int            ldC,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- C, real matrix, size m by n, modified in place.
- ldC, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{C}[j * \mathtt{ldC} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k] + (1 - \mathtt{a}) \cdot \mathtt{C}[j * \mathtt{ldC} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmexpoavg_inter_f

Computes an exponential weighted average, by element, of two matrices.

Prototype

```
void csipl_cmexpoavg_inter_f(
                    float          a,
                    void          *B,
                    int            ldB,
                    void          *C,
                    int            ldC,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- C, complex matrix, size m by n, modified in place.
- ldC, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{C}[j*\mathtt{ldC}+k] := \mathtt{a}\cdot\mathtt{B}[j*\mathtt{ldB}+k]+(1-\mathtt{a})\cdot\mathtt{C}[j*\mathtt{ldC}+k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmexpoavg_split_f

Computes an exponential weighted average, by element, of two matrices.

Prototype

```
void csipl_cmexpoavg_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    int            ldB,
                    float         *C_re,
                    float         *C_im,
                    int            ldC,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- C_re, real part of complex matrix, size m by n, modified in place.
- C_im, imaginary part of complex matrix, size m by n, modified in place.
- ldC, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{C}[j*\mathtt{ldC}+k] := \mathtt{a}\cdot\mathtt{B}[j*\mathtt{ldB}+k]+(1-\mathtt{a})\cdot\mathtt{C}[j*\mathtt{ldC}+k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vhypot_f

Computes the square root of the sum of squares, by element, of two input vectors.

Prototype

```
void csipl_vhypot_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B,
                    csipl_stride  strideB,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sqrt{(\mathtt{A}[j * \mathtt{strideA}])^2 + (\mathtt{B}[j * \mathtt{strideB}])^2}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

Intermediate overflows do not occur.

csipl_mhypot_f

Computes the square root of the sum of squares, by element, of two input matrices.

Prototype

```
void csipl_mhypot_f(
                    float        *A,
                    int           ldA,
                    float        *B,
                    int           ldB,
                    float        *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sqrt{(\mathtt{A}[j * \mathtt{ldA} + k])^2 + (\mathtt{B}[j * \mathtt{ldB} + k])^2}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

Intermediate overflows do not occur.

csipl_cvjmul_inter_f

Computes the product of a complex vector with the conjugate of a second complex vector, by element.

Prototype

```
void csipl_cvjmul_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvjmul_split_f

Computes the product of a complex vector with the conjugate of a second complex vector, by element.

Prototype

```
void csipl_cvjmul_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cmjmul_inter_f

Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.

Prototype

```
void csipl_cmjmul_inter_f(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmjmul_split_f

Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.

Prototype

```
void csipl_cmjmul_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_Dvmul_P

Computes the product, by element, of two vectors.

Prototype

```
void csipl_Dvmul_P(
                    csipl_Dvview_P *A,
                    csipl_stride    strideA,
                    csipl_Dvview_P *B,
                    csipl_stride    strideB,
                    csipl_Dvview_P *R,
                    csipl_stride    strideR,
                    csipl_length    n);
```

The following instances are supported:

- csipl_vmul_f
- csipl_vmul_i
- csipl_vmul_si
- csipl_svmul_f
- csipl_svmul_i
- csipl_svmul_si

Parameters

- A, real or complex vector, length n, input.
- strideA, integer scalar, input.
- B, real or complex vector, length n, input.
- strideB, integer scalar, input.
- R, real or complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmul_inter_f

Computes the product, by element, of two vectors.

Prototype

```
void csipl_cvmul_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmul_split_f

Computes the product, by element, of two vectors.

Prototype

```
void csipl_cvmul_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mmul_P

Computes the product, by element, of two matrices.

Prototype

```
void csipl_mmul_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

csipl_mmul_f

csipl_mmul_i

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmmul_inter_f

Computes the product, by element, of two matrices.

Prototype

```
void csipl_cmmul_inter_f(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmmul_split_f

Computes the product, by element, of two matrices.

Prototype

```
void csipl_cmmul_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rcvmul_inter_f

Computes the product, by element, of two vectors.

Prototype

```
void csipl_rcvmul_inter_f(
                    float        *A,
                    csipl_stride  strideA,
                    void         *B,
                    csipl_stride  strideB,
                    void         *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rcvmul_split_f

Computes the product, by element, of two vectors.

Prototype

```
void csipl_rcvmul_split_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rcmmul_inter_f

Computes the product, by element, of two matrices.

Prototype

```
void csipl_rcmmul_inter_f(
                    float         *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rcmmul_split_f

Computes the product, by element, of two matrices.

Prototype

```
void csipl_rcmmul_split_f(
                    float        *A,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rscvmul_inter_f

Computes the product, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvmul_inter_f(
                    float          a,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rscvmul_split_f

Computes the product, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvmul_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    csipl_stride   strideB,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_csvmul_inter_f

Computes the product, by element, of a scalar and a vector.

Prototype

```
void csipl_csvmul_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    csipl_stride     strideB,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_csvmul_split_f

Computes the product, by element, of a scalar and a vector.

Prototype

```
void csipl_csvmul_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    csipl_stride     strideB,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_smmul_f

Computes the product, by element, of a scalar and a matrix.

Prototype

```
void csipl_smmul_f(
                    float          a,
                    float         *B,
                    int            ldB,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_csmmul_inter_f

Computes the product, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmmul_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    int              ldB,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_csmmul_split_f

Computes the product, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmmul_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    int              ldB,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rscmmul_inter_f

Computes the product, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmmul_inter_f(
                    float          a,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rscmmul_split_f

Computes the product, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmmul_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    int            ldB,
                    float         *R_re,
                    float         *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vmmul_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void csipl_vmmul_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    int            ldB,
                    csipl_major    major,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real vector, input. Length n when by rows; length m when by columns.
- strideA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- major, enumerated type, input.

  CSIPL_ROW apply operation to the rows
  CSIPL_COL apply operation to the columns
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

The arguments must conform to the following:

1. `major` must be valid.

Notes

csipl_cvmmul_inter_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void csipl_cvmmul_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    int            ldB,
                    csipl_major    major,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex vector, input. Length n when by rows; length m when by columns.
- strideA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- major, enumerated type, input.

  CSIPL_ROW apply operation to the rows
  CSIPL_COL apply operation to the columns
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

The arguments must conform to the following:

1. `major` must be valid.

Notes

csipl_cvmmul_split_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void csipl_cvmmul_split_f(
                    csipl_Dvview_f *A_re,
                    csipl_Dvview_f *A_im,
                    csipl_stride    strideA,
                    csipl_Dmview_f *B_re,
                    csipl_Dmview_f *B_im,
                    int             ldB,
                    csipl_major     major,
                    csipl_Dmview_f *R_re,
                    csipl_Dmview_f *R_im,
                    int             ldR,
                    csipl_length    m,
                    csipl_length    n);
```

Parameters

- A_re, real part of real or complex vector, length p, input.
- A_im, imaginary part of real or complex vector, length p, input. Length n when by rows; length m when by columns.
- strideA, integer scalar, input.
- B_re, real part of real or complex matrix, size m by n, input.
- B_im, imaginary part of real or complex matrix, size m by n, input.
- ldB, integer scalar, input.
- major, enumerated type, input.

  CSIPL_ROW apply operation to the rows
  CSIPL_COL apply operation to the columns
- R_re, real part of real or complex matrix, size m by n, output.
- R_im, imaginary part of real or complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

The arguments must conform to the following:

1. `major` must be valid.

Notes

csipl_rvcmmul_inter_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void csipl_rvcmmul_inter_f(
                    float        *A,
                    csipl_stride  strideA,
                    void         *B,
                    int           ldB,
                    csipl_major   major,
                    void         *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real vector, input. Length n when by rows; length m when by columns.
- strideA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- major, enumerated type, input.

  CSIPL_ROW apply operation to the rows
  CSIPL_COL apply operation to the columns
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

The arguments must conform to the following:

1. `major` must be valid.

Notes

csipl_rvcmmul_split_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void csipl_rvcmmul_split_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    csipl_major   major,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real vector, input. Length n when by rows; length m when by columns.
- strideA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- major, enumerated type, input.

  CSIPL_ROW apply operation to the rows
  CSIPL_COL apply operation to the columns
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

The arguments must conform to the following:

1. `major` must be valid.

Notes

csipl_vsub_P

Computes the difference, by element, of two vectors.

Prototype

```
void csipl_vsub_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vsub_f
- csipl_vsub_i
- csipl_vsub_si

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsub_inter_f

Computes the difference, by element, of two vectors.

Prototype

```
void csipl_cvsub_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsub_split_f

Computes the difference, by element, of two vectors.

Prototype

```
void csipl_cvsub_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_msub_P

Computes the difference, by element, of two matrices.

Prototype

```
void csipl_msub_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

csipl_msub_f

csipl_msub_i

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmsub_inter_f

Computes the difference, by element, of two matrices.

Prototype

```
void csipl_cmsub_inter_f(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmsub_split_f

Computes the difference, by element, of two matrices.

Prototype

```
void csipl_cmsub_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_crmsub_inter_f

Computes the difference, by element, of two matrices.

Prototype

```
void csipl_crmsub_inter_f(
                    void          *A,
                    int            ldA,
                    float         *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_crmsub_split_f

Computes the difference, by element, of two matrices.

Prototype

```
void csipl_crmsub_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *B,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rcvsub_inter_f

Computes the difference, by element, of two vectors.

Prototype

```
void csipl_rcvsub_inter_f(
                    float        *A,
                    csipl_stride  strideA,
                    void         *B,
                    csipl_stride  strideB,
                    void         *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rcvsub_split_f

Computes the difference, by element, of two vectors.

Prototype

```
void csipl_rcvsub_split_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rcmsub_inter_f

Computes the difference, by element, of two matrices.

Prototype

```
void csipl_rcmsub_inter_f(
                    float         *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_rcmsub_split_f

Computes the difference, by element, of two matrices.

Prototype

```
void csipl_rcmsub_split_f(
                    float        *A,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_crvsub_inter_f

Computes the difference, by element, of two vectors.

Prototype

```
void csipl_crvsub_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_crvsub_split_f

Computes the difference, by element, of two vectors.

Prototype

```
void csipl_crvsub_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B,
                    csipl_stride  strideB,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_svsub_P

Computes the difference, by element, of a scalar and a vector.

Prototype

```
void csipl_svsub_P(
                    scalar_P       a,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_svsub_f
- csipl_svsub_i
- csipl_svsub_si

Parameters

- a, scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_csvsub_inter_f

Computes the difference, by element, of a scalar and a vector.

Prototype

```
void csipl_csvsub_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    csipl_stride     strideB,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \texttt{a} - \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_csvsub_split_f

Computes the difference, by element, of a scalar and a vector.

Prototype

```
void csipl_csvsub_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    csipl_stride     strideB,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_smsub_P

Computes the difference, by element, of a scalar and a matrix.

Prototype

```
void csipl_smsub_P(
                    scalar_P       a,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_smsub_f
csipl_smsub_i
```

Parameters

- a, scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_csmsub_inter_f

Computes the difference, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmsub_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    int              ldB,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_csmsub_split_f

Computes the difference, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmsub_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    int              ldB,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_smdiv_f

Computes the quotient, by element, of a scalar and a matrix.

Prototype

```
void csipl_smdiv_f(
                    float          a,
                    float         *B,
                    int            ldB,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_csmdiv_inter_f

Computes the quotient, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmdiv_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    int              ldB,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_csmdiv_split_f

Computes the quotient, by element, of a scalar and a matrix.

Prototype

```
void csipl_csmdiv_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    int              ldB,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_rscvdiv_inter_f

Computes the quotient, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvdiv_inter_f(
                    float          a,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a}/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_rscvdiv_split_f

Computes the quotient, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    csipl_stride   strideB,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] :=$ a/B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_csvdiv_inter_f

Computes the quotient, by element, of a scalar and a vector.

Prototype

```
void csipl_csvdiv_inter_f(
                    csipl_cscalar_f  a,
                    void            *B,
                    csipl_stride     strideB,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] :=$ a/B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_csvdiv_split_f

Computes the quotient, by element, of a scalar and a vector.

Prototype

```
void csipl_csvdiv_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *B_re,
                    float           *B_im,
                    csipl_stride     strideB,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}]$:= a/B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

csipl_rscvsub_inter_f

Computes the difference, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvsub_inter_f(
                    float          a,
                    void          *B,
                    csipl_stride   strideB,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \texttt{a} - \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rscvsub_split_f

Computes the difference, by element, of a real scalar and a complex vector.

Prototype

```
void csipl_rscvsub_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    csipl_stride   strideB,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_rscmdiv_inter_f

Computes the quotient, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmdiv_inter_f(
                    float          a,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

csipl_rscmdiv_split_f

Computes the quotient, by element, of a real scalar and a complex matrix.

Prototype

```
void csipl_rscmdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    int            ldB,
                    float         *R_re,
                    float         *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- a, real scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

6.4 Ternary Operations

- csipl_vam_P
- csipl_cvam_inter_P
- csipl_cvam_split_P
- csipl_vmsa_f
- csipl_cvmsa_inter_f
- csipl_cvmsa_split_f
- csipl_vmsb_f
- csipl_cvmsb_inter_f
- csipl_cvmsb_split_f
- csipl_vsam_f
- csipl_cvsam_inter_f
- csipl_cvsam_split_f
- csipl_vsbm_f
- csipl_cvsbm_inter_f
- csipl_cvsbm_split_f
- csipl_vsma_f
- csipl_cvsma_inter_f
- csipl_cvsma_split_f
- csipl_vsmsa_f
- csipl_cvsmsa_inter_f
- csipl_cvsmsa_split_f

csipl_vam_P

Computes the sum of two vectors and product of a third vector, by element.

Prototype

```
void csipl_vam_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *C,
                    csipl_stride   strideC,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

csipl_vam_f

csipl_vma_f

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- C, vector, length n, input.
- strideC, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvam_inter_P

Computes the sum of two vectors and product of a third vector, by element.

Prototype

```
void csipl_cvam_inter_P(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    void          *C,
                    csipl_stride   strideC,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cvam_inter_f
csipl_cvma_inter_f
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- C, complex vector, length n, input.
- strideC, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvam_split_P

Computes the sum of two vectors and product of a third vector, by element.

Prototype

```
void csipl_cvam_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    csipl_stride   strideB,
                    scalar_P      *C_re,
                    scalar_P      *C_im,
                    csipl_stride   strideC,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cvam_split_f
csipl_cvma_split_f
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- C_re, real part of complex vector, length n, input.
- C_im, imaginary part of complex vector, length n, input.
- strideC, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{strideA}] + \texttt{B}[j * \texttt{strideB}]) \cdot \texttt{C}[j * \texttt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vmsa_f

Computes the product of two vectors and sum of a scalar, by element.

Prototype

```
void csipl_vmsa_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    float          c,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- c, real scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}] + \mathtt{c}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmsa_inter_f

Computes the product of two vectors and sum of a scalar, by element.

Prototype

```
void csipl_cvmsa_inter_f(
                    void            *A,
                    csipl_stride     strideA,
                    void            *B,
                    csipl_stride     strideB,
                    csipl_cscalar_f  c,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- c, complex scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}] + \mathtt{c}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmsa_split_f

Computes the product of two vectors and sum of a scalar, by element.

Prototype

```
void csipl_cvmsa_split_f(
                    float           *A_re,
                    float           *A_im,
                    csipl_stride     strideA,
                    float           *B_re,
                    float           *B_im,
                    csipl_stride     strideB,
                    csipl_cscalar_f  c_re,
                    csipl_cscalar_f  c_im,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- c_re, real part of complex scalar, input.
- c_im, imaginary part of complex scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}] \cdot \texttt{B}[j * \texttt{strideB}] + \texttt{c}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vmsb_f

Computes the product of two vectors and difference of a third vector, by element.

Prototype

```
void csipl_vmsb_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    float         *C,
                    csipl_stride   strideC,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- C, real vector, length n, input.
- strideC, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}] - \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmsb_inter_f

Computes the product of two vectors and difference of a third vector, by element.

Prototype

```
void csipl_cvmsb_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    void          *C,
                    csipl_stride   strideC,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- C, complex vector, length n, input.
- strideC, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}] - \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvmsb_split_f

Computes the product of two vectors and difference of a third vector, by element.

Prototype

```
void csipl_cvmsb_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *C_re,
                    float        *C_im,
                    csipl_stride  strideC,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- C_re, real part of complex vector, length n, input.
- C_im, imaginary part of complex vector, length n, input.
- strideC, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}] \cdot \texttt{B}[j * \texttt{strideB}] - \texttt{C}[j * \texttt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vsam_f

Computes the sum of a vector and a scalar, and product with a second vector, by element.

Prototype

```
void csipl_vsam_f(
                    float         *A,
                    csipl_stride   strideA,
                    float          b,
                    float         *C,
                    csipl_stride   strideC,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- b, real scalar, input.
- C, real vector, length n, input.
- strideC, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] + \mathtt{b}) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsam_inter_f

Computes the sum of a vector and a scalar, and product with a second vector, by element.

Prototype

```
void csipl_cvsam_inter_f(
                    void            *A,
                    csipl_stride     strideA,
                    csipl_cscalar_f  b,
                    void            *C,
                    csipl_stride     strideC,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- b, complex scalar, input.
- C, complex vector, length n, input.
- strideC, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] + \mathtt{b}) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsam_split_f

Computes the sum of a vector and a scalar, and product with a second vector, by element.

Prototype

```
void csipl_cvsam_split_f(
                    float           *A_re,
                    float           *A_im,
                    csipl_stride     strideA,
                    csipl_cscalar_f  b_re,
                    csipl_cscalar_f  b_im,
                    float           *C_re,
                    float           *C_im,
                    csipl_stride     strideC,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- b_re, real part of complex scalar, input.
- b_im, imaginary part of complex scalar, input.
- C_re, real part of complex vector, length n, input.
- C_im, imaginary part of complex vector, length n, input.
- strideC, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{strideA}] + \texttt{b}) \cdot \texttt{C}[j * \texttt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vsbm_f

Computes the difference of two vectors, and product with a third vector, by element.

Prototype

```
void csipl_vsbm_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B,
                    csipl_stride  strideB,
                    float        *C,
                    csipl_stride  strideC,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- C, real vector, length n, input.
- strideC, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsbm_inter_f

Computes the difference of two vectors, and product with a third vector, by element.

Prototype

```
void csipl_cvsbm_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    void          *C,
                    csipl_stride   strideC,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- C, complex vector, length n, input.
- strideC, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsbm_split_f

Computes the difference of two vectors, and product with a third vector, by element.

Prototype

```
void csipl_cvsbm_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *C_re,
                    float        *C_im,
                    csipl_stride  strideC,
                    float        *R_re,
                    float        *R_im,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- C_re, real part of complex vector, length n, input.
- C_im, imaginary part of complex vector, length n, input.
- strideC, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{strideA}] - \texttt{B}[j * \texttt{strideB}]) \cdot \texttt{C}[j * \texttt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vsma_f

Computes the product of a vector and a scalar, and sum with a second vector, by element.

Prototype

```
void csipl_vsma_f(
                    float         *A,
                    csipl_stride   strideA,
                    float          b,
                    float         *C,
                    csipl_stride   strideC,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- b, real scalar, input.
- C, real vector, length n, input.
- strideC, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{b} + \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsma_inter_f

Computes the product of a vector and a scalar, and sum with a second vector, by element.

Prototype

```
void csipl_cvsma_inter_f(
                    void            *A,
                    csipl_stride     strideA,
                    csipl_cscalar_f  b,
                    void            *C,
                    csipl_stride     strideC,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- b, complex scalar, input.
- C, complex vector, length n, input.
- strideC, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{b} + \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsma_split_f

Computes the product of a vector and a scalar, and sum with a second vector, by element.

Prototype

```
void csipl_cvsma_split_f(
                    float           *A_re,
                    float           *A_im,
                    csipl_stride     strideA,
                    csipl_cscalar_f  b_re,
                    csipl_cscalar_f  b_im,
                    float           *C_re,
                    float           *C_im,
                    csipl_stride     strideC,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- b_re, real part of complex scalar, input.
- b_im, imaginary part of complex scalar, input.
- C_re, real part of complex vector, length n, input.
- C_im, imaginary part of complex vector, length n, input.
- strideC, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}] \cdot \texttt{b} + \texttt{C}[j * \texttt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vsmsa_f

Computes the product of a vector and a scalar, and sum with a second scalar, by element.

Prototype

```
void csipl_vsmsa_f(
                    float         *A,
                    csipl_stride   strideA,
                    float          b,
                    float          c,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- b, real scalar, input.
- c, real scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{b} + \mathtt{c}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsmsa_inter_f

Computes the product of a vector and a scalar, and sum with a second scalar, by element.

Prototype

```
void csipl_cvsmsa_inter_f(
                    void            *A,
                    csipl_stride     strideA,
                    csipl_cscalar_f  b,
                    csipl_cscalar_f  c,
                    void            *R,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- b, complex scalar, input.
- c, complex scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{b} + \mathtt{c}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvsmsa_split_f

Computes the product of a vector and a scalar, and sum with a second scalar, by element.

Prototype

```
void csipl_cvsmsa_split_f(
                    float           *A_re,
                    float           *A_im,
                    csipl_stride     strideA,
                    csipl_cscalar_f  b_re,
                    csipl_cscalar_f  b_im,
                    csipl_cscalar_f  c_re,
                    csipl_cscalar_f  c_im,
                    float           *R_re,
                    float           *R_im,
                    csipl_stride     strideR,
                    csipl_length     n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- b_re, real part of complex scalar, input.
- b_im, imaginary part of complex scalar, input.
- c_re, real part of complex scalar, input.
- c_im, imaginary part of complex scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{b} + \mathtt{c}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

6.5 Logical Operations

- csipl_valltrue_bl
- csipl_malltrue_bl
- csipl_vanytrue_bl
- csipl_manytrue_bl
- csipl_vleq_P
- csipl_cvleq_inter_P
- csipl_cvleq_split_P
- csipl_mleq_P
- csipl_cmleq_inter_P
- csipl_cmleq_split_P
- csipl_vlge_P
- csipl_mlge_P
- csipl_vlgt_P
- csipl_mlgt_P
- csipl_vlle_P
- csipl_mlle_P
- csipl_vllt_P
- csipl_mllt_P
- csipl_vlne_P
- csipl_cvlne_inter_P
- csipl_cvlne_split_P
- csipl_mlne_P
- csipl_cmlne_inter_P
- csipl_cmlne_split_P

csipl_valltrue_bl

Returns true if all the elements of a vector are true.

Prototype

```
csipl_scalar_bl csipl_valltrue_bl(
                    signed int    *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, boolean vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- boolean scalar.

Description

return value := (for all elements $\mathtt{A}[j * \mathtt{strideA}] = \mathtt{true}$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_malltrue_bl

Returns true if all the elements of a vector are true.

Prototype

```
csipl_scalar_bl csipl_malltrue_bl(
                    signed int    *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, boolean vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- boolean scalar.

Description

return value := (for all elements A[j * strideA] = true) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vanytrue_bl

Returns true if one or more elements of a vector are true.

Prototype

```
csipl_scalar_bl csipl_vanytrue_bl(
                    signed int    *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, boolean vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- boolean scalar.

Description

return value := (for any element A[$j *$ strideA] = true) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The logical complement of any true is none true.

csipl_manytrue_bl

Returns true if one or more elements of a vector are true.

Prototype

```
csipl_scalar_bl csipl_manytrue_bl(
                    signed int    *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, boolean vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- boolean scalar.

Description

return value := (for any element A[$j *$ strideA] = true) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The logical complement of any true is none true.

csipl_vleq_P

Computes the boolean comparison of 'equal', by element, of two vectors.

Prototype

```
void csipl_vleq_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vleq_f
- csipl_vleq_i

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] = \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cvleq_inter_P

Computes the boolean comparison of 'equal', by element, of two vectors.

Prototype

```
void csipl_cvleq_inter_P(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cvleq_inter_f
csipl_cvleq_inter_i
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] = \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cvleq_split_P

Computes the boolean comparison of 'equal', by element, of two vectors.

Prototype

```
void csipl_cvleq_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    csipl_stride   strideB,
                    signed int    *R_re,
                    signed int    *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cvleq_split_f
csipl_cvleq_split_i
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of boolean vector, length n, output.
- R_im, imaginary part of boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{strideA}] = \texttt{B}[j * \texttt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_mleq_P

Computes the boolean comparison of 'equal', by element, of two vectors/matrices.

Prototype

```
void csipl_mleq_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

csipl_mleq_f

csipl_mleq_i

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] = \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cmleq_inter_P

Computes the boolean comparison of 'equal', by element, of two vectors/matrices.

Prototype

```
void csipl_cmleq_inter_P(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cmleq_inter_f
csipl_cmleq_inter_i
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] = \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cmleq_split_P

Computes the boolean comparison of 'equal', by element, of two vectors/matrices.

Prototype

```
void csipl_cmleq_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    int            ldB,
                    signed int    *R_re,
                    signed int    *R_im,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cmleq_split_f
csipl_cmleq_split_i
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, input.
- B_im, imaginary part of complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R_re, real part of boolean vector, length n, output.
- R_im, imaginary part of boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{ldA} + k] = \texttt{B}[j * \texttt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_vlge_P

Computes the boolean comparison of 'greater than or equal', by element, of two vectors.

Prototype

```
void csipl_vlge_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vlge_f
- csipl_vlge_i

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] \geqslant \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_mlge_P

Computes the boolean comparison of 'greater than or equal', by element, of two vectors/matrices.

Prototype

```
void csipl_mlge_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

`csipl_mlge_f`

`csipl_mlge_i`

Parameters

- `A`, matrix, size m by n, input.
- `ldA`, integer scalar, input.
- `B`, matrix, size m by n, input.
- `ldB`, integer scalar, input.
- `R`, boolean vector, length n, output.
- `strideR`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{ldA} + k] \geqslant \texttt{B}[j * \texttt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_vlgt_P

Computes the boolean comparison of 'greater than', by element, of two vectors.

Prototype

```
void csipl_vlgt_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_vlgt_f
csipl_vlgt_i
```

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] > \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_mlgt_P

Computes the boolean comparison of 'greater than', by element, of two vectors/matrices.

Prototype

```
void csipl_mlgt_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_mlgt_f
csipl_mlgt_i
```

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] > \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_vlle_P

Computes the boolean comparison of 'less than or equal', by element, of two vectors.

Prototype

```
void csipl_vlle_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_vlle_f
csipl_vlle_i
```

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] \leqslant \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_mlle_P

Computes the boolean comparison of 'less than or equal', by element, of two vectors/matrices.

Prototype

```
void csipl_mlle_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mlle_f
- csipl_mlle_i

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] \leqslant \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_vllt_P

Computes the boolean comparison of 'less than', by element, of two vectors.

Prototype

```
void csipl_vllt_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_vllt_f
csipl_vllt_i
```

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] < \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_mllt_P

Computes the boolean comparison of 'less than', by element, of two vectors/matrices.

Prototype

```
void csipl_mllt_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

csipl_mllt_f

csipl_mllt_i

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] < \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_vlne_P

Computes the boolean comparison of 'not equal', by element, of two vectors.

Prototype

```
void csipl_vlne_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- `csipl_vlne_f`
- `csipl_vlne_i`

Parameters

- `A`, vector, length n, input.
- `strideA`, integer scalar, input.
- `B`, vector, length n, input.
- `strideB`, integer scalar, input.
- `R`, boolean vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] \neq \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cvlne_inter_P

Computes the boolean comparison of 'not equal', by element, of two vectors.

Prototype

```
void csipl_cvlne_inter_P(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

`csipl_cvlne_inter_f`

`csipl_cvlne_inter_i`

Parameters

- `A`, complex vector, length n, input.
- `strideA`, integer scalar, input.
- `B`, complex vector, length n, input.
- `strideB`, integer scalar, input.
- `R`, boolean vector, length n, output.
- `strideR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] \neq \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cvlne_split_P

Computes the boolean comparison of 'not equal', by element, of two vectors.

Prototype

```
void csipl_cvlne_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    csipl_stride   strideA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    csipl_stride   strideB,
                    signed int    *R_re,
                    signed int    *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cvlne_split_f
csipl_cvlne_split_i
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of boolean vector, length n, output.
- R_im, imaginary part of boolean vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{strideA}] \neq \texttt{B}[j * \texttt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_mlne_P

Computes the boolean comparison of 'not equal', by element, of two vectors/matrices.

Prototype

```
void csipl_mlne_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

csipl_mlne_f

csipl_mlne_i

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] \neq \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cmlne_inter_P

Computes the boolean comparison of 'not equal', by element, of two vectors/matrices.

Prototype

```
void csipl_cmlne_inter_P(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    signed int    *R,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cmlne_inter_f
csipl_cmlne_inter_i
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, input.
- ldB, integer scalar, input.
- R, boolean vector, length n, output.
- strideR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] \neq \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

csipl_cmlne_split_P

Computes the boolean comparison of 'not equal', by element, of two vectors/matrices.

Prototype

```
void csipl_cmlne_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    int            ldB,
                    signed int    *R_re,
                    signed int    *R_im,
                    csipl_stride   strideR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

`csipl_cmlne_split_f`

`csipl_cmlne_split_i`

Parameters

- `A_re`, real part of complex matrix, size m by n, input.
- `A_im`, imaginary part of complex matrix, size m by n, input.
- `ldA`, integer scalar, input.
- `B_re`, real part of complex matrix, size m by n, input.
- `B_im`, imaginary part of complex matrix, size m by n, input.
- `ldB`, integer scalar, input.
- `R_re`, real part of boolean vector, length n, output.
- `R_im`, imaginary part of boolean vector, length n, output.
- `strideR`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{ldA} + k] \neq \texttt{B}[j * \texttt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

6.6 Selection Operations

- csipl_vclip_P
- csipl_mclip_f
- csipl_vinvclip_P
- csipl_minvclip_f
- csipl_vindexbool
- csipl_vmax_f
- csipl_mmax_f
- csipl_vmaxmg_f
- csipl_mmaxmg_f
- csipl_vcmaxmgsq_inter_f
- csipl_vcmaxmgsq_split_f
- csipl_vcmaxmgsqval_inter_f
- csipl_vcmaxmgsqval_split_f
- csipl_vmaxmgval_f
- csipl_mmaxmgval_f
- csipl_vmaxval_f
- csipl_mmaxval_f
- csipl_vmin_f
- csipl_mmin_f
- csipl_vminmg_f
- csipl_mminmg_f
- csipl_vcminmgsq_inter_f
- csipl_vcminmgsq_split_f
- csipl_vcminmgsqval_inter_f
- csipl_vcminmgsqval_split_f
- csipl_vminmgval_f
- csipl_mminmgval_f
- csipl_vminval_f
- csipl_mminval_f

csipl_vclip_P

Computes the generalised double clip, by element, of two vectors.

Prototype

```
void csipl_vclip_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P       t1,
                    scalar_P       t2,
                    scalar_P       c1,
                    scalar_P       c2,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vclip_f
- csipl_vclip_i
- csipl_vclip_si

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- t1, scalar, input.
- t2, scalar, input.
- c1, scalar, input.
- c2, scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

If A[j * strideA] $\leqslant$ t1 then R[j * strideR] := c1 else if A[j * strideA] $<$ t2 then R[j * strideR] := A[j * strideA] else R[j * strideR] := c2.

Restrictions

Errors

Notes

The clipping rules are evaluated sequentially: once a rule is met, the following rules are ignored. The threshold variables are unrestricted and need not be in increasing order.

csipl_mclip_f

Computes the generalised double clip, by element, of two matrices.

Prototype

```
void csipl_mclip_f(
                    float         *A,
                    int            ldA,
                    float          t1,
                    float          t2,
                    float          c1,
                    float          c2,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- t1, real scalar, input.
- t2, real scalar, input.
- c1, real scalar, input.
- c2, real scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

If $\mathtt{A}[j*\mathtt{ldA}+k] \leqslant \mathtt{t1}$ then $\mathtt{R}[j*\mathtt{ldR}+k] := \mathtt{c1}$ else if $\mathtt{A}[j*\mathtt{ldA}+k] < \mathtt{t2}$ then $\mathtt{R}[j*\mathtt{ldR}+k] := \mathtt{A}[j*\mathtt{ldA}+k]$ else $\mathtt{R}[j*\mathtt{ldR}+k] := \mathtt{c2}$.

Restrictions

Errors

Notes

The clipping rules are evaluated sequentially: once a rule is met, the following rules are ignored. The threshold variables are unrestricted and need not be in increasing order.

csipl_vinvclip_P

Computes the generalised inverted double clip, by element, of two vectors.

Prototype

```
void csipl_vinvclip_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P       t1,
                    scalar_P       t2,
                    scalar_P       t3,
                    scalar_P       c1,
                    scalar_P       c2,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vinvclip_f
- csipl_vinvclip_i
- csipl_vinvclip_si

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- t1, scalar, input.
- t2, scalar, input.
- t3, scalar, input.
- c1, scalar, input.
- c2, scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

If $\texttt{A}[j * \texttt{strideA}] < \texttt{t1}$ then $\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}]$ else if $\texttt{A}[j * \texttt{strideA}] < \texttt{t2}$ then $\texttt{R}[j * \texttt{strideR}] := \texttt{c1}$ else if $\texttt{A}[j * \texttt{strideA}] \leqslant \texttt{t3}$ then $\texttt{R}[j * \texttt{strideR}] := \texttt{c2}$ else $\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}]$.

Restrictions

Errors

Notes

The clipping rules are evaluated sequentially: once a rule is met, the following rules are ignored. The threshold variables are unrestricted and need not be in increasing order.

csipl_minvclip_f

Computes the generalised inverted double clip, by element, of two matrices.

Prototype

```
void csipl_minvclip_f(
                    float         *A,
                    int            ldA,
                    float          t1,
                    float          t2,
                    float          t3,
                    float          c1,
                    float          c2,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- t1, real scalar, input.
- t2, real scalar, input.
- t3, real scalar, input.
- c1, real scalar, input.
- c2, real scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

If $\mathtt{A}[j * \mathtt{ldA} + k] < \mathtt{t1}$ then $\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ else if $\mathtt{A}[j * \mathtt{ldA} + k] < \mathtt{t2}$ then $\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{c1}$ else if $\mathtt{A}[j * \mathtt{ldA} + k] \leqslant \mathtt{t3}$ then $\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{c2}$ else $\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$.

Restrictions

Errors

Notes

The clipping rules are evaluated sequentially: once a rule is met, the following rules are ignored. The threshold variables are unrestricted and need not be in increasing order.

csipl_vindexbool

Computes an index vector of the indices of the non-false elements of the boolean vector, and returns the number of non-false elements.

Prototype

```
unsigned int  csipl_vindexbool(
                    signed int     *X,
                    csipl_stride    strideX,
                    unsigned int   *Y,
                    csipl_stride    strideY,
                    csipl_length    n);
```

Parameters

- X, boolean vector, length n, input.
- strideX, integer scalar, input.
- Y, vector-index vector, length n, output.
- strideY, integer scalar, input.
- n, integer scalar, input.

Return Value

- integer scalar.

Description

Returns an index vector Y of the indices of the non-false elements of the boolean vector X. The index vector is ordered: lower indices appear before higher indices. The return value is the number of non-false elements.

Restrictions

No in-place operations are allowed.

Errors

Notes

csipl_vmax_f

Computes the maximum, by element, of two vectors.

Prototype

```
void csipl_vmax_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B,
                    csipl_stride  strideB,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \max\{\mathtt{A}[j * \mathtt{strideA}], \mathtt{B}[j * \mathtt{strideB}]\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mmax_f

Computes the maximum, by element, of two matrices.

Prototype

```
void csipl_mmax_f(
                    float         *A,
                    int            ldA,
                    float         *B,
                    int            ldB,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \max\{\mathtt{A}[j * \mathtt{ldA} + k], \mathtt{B}[j * \mathtt{ldB} + k]\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vmaxmg_f

Computes the maximum magnitude (absolute value), by element, of two vectors.

Prototype

```
void csipl_vmaxmg_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \max\{|\mathtt{A}[j * \mathtt{strideA}]|, |\mathtt{B}[j * \mathtt{strideB}]|\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mmaxmg_f

Computes the maximum magnitude (absolute value), by element, of two matrices.

Prototype

```
void csipl_mmaxmg_f(
                    float         *A,
                    int            ldA,
                    float         *B,
                    int            ldB,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \max\{|\mathtt{A}[j * \mathtt{ldA} + k]|, |\mathtt{B}[j * \mathtt{ldB} + k]|\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vcmaxmgsq_inter_f

Computes the maximum magnitude squared, by element, of two complex vectors.

Prototype

```
void csipl_vcmaxmgsq_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \max\{|\mathtt{A}[j * \mathtt{strideA}]|^2, |\mathtt{B}[j * \mathtt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vcmaxmgsq_split_f

Computes the maximum magnitude squared, by element, of two complex vectors.

Prototype

```
void csipl_vcmaxmgsq_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \max\{|\mathtt{A}[j * \mathtt{strideA}]|^2, |\mathtt{B}[j * \mathtt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vcmaxmgsqval_inter_f

Returns the index and value of the maximum magnitude squared of the elements of a complex vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vcmaxmgsqval_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    csipl_index   *index,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \max\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum magnitude squared values, the index of the first maximum magnitude squared is returned in the index.

csipl_vcmaxmgsqval_split_f

Returns the index and value of the maximum magnitude squared of the elements of a complex vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vcmaxmgsqval_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    csipl_index  *index,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \max\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum magnitude squared values, the index of the first maximum magnitude squared is returned in the index.

csipl_vmaxmgval_f

Returns the index and value of the maximum absolute value of the elements of a vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vmaxmgval_f(
                    float         *A,
                    csipl_stride   strideA,
                    csipl_index   *index,
                    csipl_length   n);
```

Parameters

- `A`, real vector, length n, input.
- `strideA`, integer scalar, input.
- `index`, pointer to vector-index scalar, output.
- `n`, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \max\{|\mathtt{A}[j * \mathtt{strideA}]|\}$ where $0 \leqslant j < n$.

If `index` is not `NULL` the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum absolute values, the index of the first maximum absolute value is returned in the index.

csipl_mmaxmgval_f

Returns the index and value of the maximum absolute value of the elements of a matrix. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_mmaxmgval_f(
                    float         *A,
                    int            ldA,
                    csipl_index   *index,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \max\{|\mathtt{A}[j * \mathtt{ldA} + k]|\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the matrix has more than one element with identical maximum absolute values, the index of the first maximum absolute value is returned in the index.

csipl_vmaxval_f

Returns the index and value of the maximum value of the elements of a vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vmaxval_f(
                    float         *A,
                    csipl_stride   strideA,
                    csipl_index   *index,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value := $\max\{\mathtt{A}[j * \mathtt{strideA}]\}$ where $0 \leqslant j < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum values the index of the first maximum is returned in the index.

csipl_mmaxval_f

Returns the index and value of the maximum value of the elements of a matrix. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_mmaxval_f(
                    float        *A,
                    int           ldA,
                    csipl_index  *index,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value := $\max\{\mathtt{A}[j * \mathtt{ldA} + k]\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the matrix has more than one element with identical maximum values the index of the first maximum is returned in the index.

csipl_vmin_f

Computes the minimum, by element, of two vectors.

Prototype

```
void csipl_vmin_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{\mathtt{A}[j * \mathtt{strideA}], \mathtt{B}[j * \mathtt{strideB}]\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mmin_f

Computes the minimum, by element, of two matrices.

Prototype

```
void csipl_mmin_f(
                    float        *A,
                    int           ldA,
                    float        *B,
                    int           ldB,
                    float        *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \min\{\mathtt{A}[j * \mathtt{ldA} + k], \mathtt{B}[j * \mathtt{ldB} + k]\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vminmg_f

Computes the minimum magnitude (absolute value), by element, of two vectors.

Prototype

```
void csipl_vminmg_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{|\mathtt{A}[j * \mathtt{strideA}]|, |\mathtt{B}[j * \mathtt{strideB}]|\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mminmg_f

Computes the minimum magnitude (absolute value), by element, of two matrices.

Prototype

```
void csipl_mminmg_f(
                    float        *A,
                    int           ldA,
                    float        *B,
                    int           ldB,
                    float        *R,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \min\{|\mathtt{A}[j * \mathtt{ldA} + k]|, |\mathtt{B}[j * \mathtt{ldB} + k]|\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vcminmgsq_inter_f

Computes the minimum magnitude squared, by element, of two complex vectors.

Prototype

```
void csipl_vcminmgsq_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{|\mathtt{A}[j * \mathtt{strideA}]|^2, |\mathtt{B}[j * \mathtt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vcminmgsq_split_f

Computes the minimum magnitude squared, by element, of two complex vectors.

Prototype

```
void csipl_vcminmgsq_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{|\mathtt{A}[j * \mathtt{strideA}]|^2, |\mathtt{B}[j * \mathtt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vcminmgsqval_inter_f

Returns the index and value of the minimum magnitude squared of the elements of a complex vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vcminmgsqval_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    csipl_index   *index,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \min\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum magnitude squared values, the index of the first minimum magnitude squared is returned in the index.

csipl_vcminmgsqval_split_f

Returns the index and value of the minimum magnitude squared of the elements of a complex vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vcminmgsqval_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    csipl_index  *index,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \min\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum magnitude squared values, the index of the first minimum magnitude squared is returned in the index.

csipl_vminmgval_f

Returns the index and value of the minimum absolute value of the elements of a vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vminmgval_f(
                    float        *A,
                    csipl_stride  strideA,
                    csipl_index  *index,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value := $\min\{|\mathtt{A}[j * \mathtt{strideA}]|\}$ where $0 \leqslant j < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum absolute value values, the index of the first minimum absolute value is returned in the index.

csipl_mminmgval_f

Returns the index and value of the minimum absolute value of the elements of a matrix. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_mminmgval_f(
                    float        *A,
                    int           ldA,
                    csipl_index  *index,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \min\{|\mathtt{A}[j * \mathtt{ldA} + k]|\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the matrix has more than one element with identical minimum absolute value values, the index of the first minimum absolute value is returned in the index.

csipl_vminval_f

Returns the index and value of the minimum value of the elements of a vector. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_vminval_f(
                    float        *A,
                    csipl_stride  strideA,
                    csipl_index  *index,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value := $\min\{A[j * \texttt{strideA}]\}$ where $0 \leqslant j < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum values the index of the first minimum is returned in the index.

csipl_mminval_f

Returns the index and value of the minimum value of the elements of a matrix. The index is returned by reference as one of the arguments.

Prototype

```
float  csipl_mminval_f(
                    float        *A,
                    int           ldA,
                    csipl_index  *index,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- index, pointer to vector-index scalar, output.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \min\{\mathtt{A}[j * \mathtt{ldA} + k]\}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the matrix has more than one element with identical minimum values the index of the first minimum is returned in the index.

6.7 Bitwise and Boolean Logical Operators

- csipl_vand_P
- csipl_mand_P
- csipl_vnot_P
- csipl_mnot_P
- csipl_vor_P
- csipl_mor_P
- csipl_vxor_P
- csipl_mxor_P

csipl_vand_P

Computes the bitwise and, by element, of two vectors.

Prototype

```
void csipl_vand_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vand_i
- csipl_vand_si
- csipl_vand_bl

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ and $\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mand_P

Computes the bitwise and, by element, of two matrices.

Prototype

```
void csipl_mand_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mand_i
- csipl_mand_si
- csipl_mand_bl

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ and $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vnot_P

Computes the bitwise not (one's complement), by element, of two vectors.

Prototype

```
void csipl_vnot_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vnot_i
- csipl_vnot_si
- csipl_vnot_bl

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \text{not}(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mnot_P

Computes the bitwise not (one's complement), by element, of two matrices.

Prototype

```
void csipl_mnot_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mnot_i
- csipl_mnot_si
- csipl_mnot_bl

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \text{not}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vor_P

Computes the bitwise inclusive or, by element, of two vectors.

Prototype

```
void csipl_vor_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vor_i
- csipl_vor_si
- csipl_vor_bl

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}]$:= A$[j * \texttt{strideA}]$ or B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mor_P

Computes the bitwise inclusive or, by element, of two matrices.

Prototype

```
void csipl_mor_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mor_i
- csipl_mor_si
- csipl_mor_bl

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ or $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vxor_P

Computes the bitwise exclusive or, by element, of two vectors.

Prototype

```
void csipl_vxor_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vxor_i
- csipl_vxor_si
- csipl_vxor_bl

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- B, vector, length n, input.
- strideB, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ xor $\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mxor_P

Computes the bitwise exclusive or, by element, of two matrices.

Prototype

```
void csipl_mxor_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mxor_i
- csipl_mxor_si
- csipl_mxor_bl

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- B, matrix, size m by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ xor $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

6.8 Element Generation and Copy

- csipl_vcopy_P_P
- csipl_cvcopy_inter_f_f
- csipl_cvcopy_split_f_f
- csipl_mcopy_P_P
- csipl_cmcopy_inter_f_f
- csipl_cmcopy_split_f_f
- csipl_vfill_P
- csipl_cvfill_inter_f
- csipl_cvfill_split_f
- csipl_mfill_P
- csipl_cmfill_inter_f
- csipl_cmfill_split_f
- csipl_vramp_P

csipl_vcopy_P_P

Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void csipl_vcopy_P_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vcopy_f_f
- csipl_vcopy_f_i
- csipl_vcopy_f_si
- csipl_vcopy_f_bl
- csipl_vcopy_i_f
- csipl_vcopy_i_i
- csipl_vcopy_i_si
- csipl_vcopy_i_vi
- csipl_vcopy_si_f
- csipl_vcopy_si_i
- csipl_vcopy_si_si
- csipl_vcopy_bl_f
- csipl_vcopy_bl_bl
- csipl_vcopy_vi_i
- csipl_vcopy_vi_vi
- csipl_vcopy_mi_mi

Parameters

- A, vector, length n, input.
- strideA, integer scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

csipl_cvcopy_inter_f_f

Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void csipl_cvcopy_inter_f_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

csipl_cvcopy_split_f_f

Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void csipl_cvcopy_split_f_f(
                    float         *A_re,
                    float         *A_im,
                    csipl_stride   strideA,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

csipl_mcopy_P_P

Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void csipl_mcopy_P_P(
                     scalar_P      *A,
                     int            ldA,
                     scalar_P      *R,
                     int            ldR,
                     csipl_length   m,
                     csipl_length   n);
```

The following instances are supported:

```
csipl_mcopy_f_f
csipl_mcopy_f_i
csipl_mcopy_f_si
csipl_mcopy_f_bl
csipl_mcopy_i_f
csipl_mcopy_i_i
csipl_mcopy_i_si
csipl_mcopy_si_f
csipl_mcopy_si_i
csipl_mcopy_si_si
csipl_mcopy_bl_f
csipl_mcopy_bl_bl
```

Parameters

- `A`, matrix, size m by n, input.
- `ldA`, integer scalar, input.
- `R`, matrix, size m by n, output.
- `ldR`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\texttt{R}[j * \texttt{ldR} + k] := \texttt{A}[j * \texttt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

csipl_cmcopy_inter_f_f

Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void csipl_cmcopy_inter_f_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

csipl_cmcopy_split_f_f

Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void csipl_cmcopy_split_f_f(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

csipl_vfill_P

Fill a vector with a constant value.

Prototype

```
void csipl_vfill_P(
                    scalar_P       a,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vfill_f
- csipl_vfill_i
- csipl_vfill_si

Parameters

- a, scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] :=$ a where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvfill_inter_f

Fill a vector with a constant value.

Prototype

```
void csipl_cvfill_inter_f(
                    csipl_cscalar_f   a,
                    void             *R,
                    csipl_stride      strideR,
                    csipl_length      n);
```

Parameters

- a, complex scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] :=$ a where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_cvfill_split_f

Fill a vector with a constant value.

Prototype

```
void csipl_cvfill_split_f(
                    csipl_cscalar_f   a_re,
                    csipl_cscalar_f   a_im,
                    float            *R_re,
                    float            *R_im,
                    csipl_stride      strideR,
                    csipl_length      n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] :=$ a where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mfill_P

Fill a matrix with a constant value.

Prototype

```
void csipl_mfill_P(
                    scalar_P       a,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mfill_f
- csipl_mfill_i
- csipl_mfill_si

Parameters

- a, scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{ldR} + k] :=$ a where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmfill_inter_f

Fill a matrix with a constant value.

Prototype

```
void csipl_cmfill_inter_f(
                    csipl_cscalar_f  a,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a, complex scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{ldR} + k] :=$ a where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_cmfill_split_f

Fill a matrix with a constant value.

Prototype

```
void csipl_cmfill_split_f(
                    csipl_cscalar_f  a_re,
                    csipl_cscalar_f  a_im,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- a_re, real part of complex scalar, input.
- a_im, imaginary part of complex scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{ldR} + k] :=$ a where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vramp_P

Computes a vector ramp by starting at an initial value and incrementing each successive element by the ramp step size.

Prototype

```
void csipl_vramp_P(
                    scalar_P       alpha,
                    scalar_P       beta,
                    scalar_P      *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vramp_f
- csipl_vramp_i
- csipl_vramp_si

Parameters

- alpha, scalar, input.
- beta, scalar, input.
- R, vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j]$:= alpha $+ j \cdot$ beta where $0 \leqslant j < n$.

Restrictions

Errors

Notes

6.9 Manipulation Operations

- csipl_vcmplx_inter_f
- csipl_vcmplx_split_f
- csipl_mcmplx_f
- csipl_vgather_P
- csipl_cvgather_inter_f
- csipl_cvgather_split_f
- csipl_mgather_P
- csipl_cmgather_inter_f
- csipl_cmgather_split_f
- csipl_vimag_inter_f
- csipl_vimag_split_f
- csipl_mimag_f
- csipl_vpolar_inter_f
- csipl_vpolar_split_f
- csipl_mpolar_inter_f
- csipl_mpolar_split_f
- csipl_vreal_inter_f
- csipl_vreal_split_f
- csipl_mreal_f
- csipl_vrect_inter_f
- csipl_vrect_split_f
- csipl_mrect_inter_f
- csipl_mrect_split_f
- csipl_vscatter_P
- csipl_cvscatter_inter_f
- csipl_cvscatter_split_f
- csipl_mscatter_P
- csipl_cmscatter_inter_f
- csipl_cmscatter_split_f
- csipl_vswap_P
- csipl_cvswap_inter_f
- csipl_cvswap_split_f
- csipl_mswap_P

- csipl_cmswap_inter_f
- csipl_cmswap_split_f

csipl_vcmplx_inter_f

Form a complex vector from two real vectors.

Prototype

```
void csipl_vcmplx_inter_f(
                    float        *A,
                    csipl_stride  strideA,
                    float        *B,
                    csipl_stride  strideB,
                    void         *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R, complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathrm{i} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vcmplx_split_f

Form a complex vector from two real vectors.

Prototype

```
void csipl_vcmplx_split_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    float         *R_re,
                    float         *R_im,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- R_re, real part of complex vector, length n, output.
- R_im, imaginary part of complex vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathrm{i} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mcmplx_f

Form a complex matrix from two real matrices.

Prototype

```
void csipl_mcmplx_f(
                    float         *A,
                    int            ldA,
                    float         *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, real matrix, size m by n, input.
- ldA, integer scalar, input.
- B, real matrix, size m by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathrm{i} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vgather_P

The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void csipl_vgather_P(
                    scalar_P       *X,
                    csipl_stride    strideX,
                    unsigned int   *I,
                    csipl_stride    strideI,
                    scalar_P       *Y,
                    csipl_stride    strideY,
                    csipl_length    n);
```

The following instances are supported:

```
csipl_vgather_f
csipl_vgather_i
csipl_vgather_si
```

Parameters

- X, vector, length m, input.
- strideX, integer scalar, input.
- I, vector-index vector, length n, input.
- strideI, integer scalar, input.
- Y, vector, length n, output.
- strideY, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Y$[j]$:= X[I$[j]$] where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

csipl_cvgather_inter_f

The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void csipl_cvgather_inter_f(
                    void           *X,
                    csipl_stride    strideX,
                    unsigned int   *I,
                    csipl_stride    strideI,
                    void           *Y,
                    csipl_stride    strideY,
                    csipl_length    n);
```

Parameters

- X, complex vector, length m, input.
- strideX, integer scalar, input.
- I, vector-index vector, length n, input.
- strideI, integer scalar, input.
- Y, complex vector, length n, output.
- strideY, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[j] := \mathtt{X}[\mathtt{I}[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

csipl_cvgather_split_f

The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void csipl_cvgather_split_f(
                    float          *X_re,
                    float          *X_im,
                    csipl_stride    strideX,
                    unsigned int   *I_re,
                    unsigned int   *I_im,
                    csipl_stride    strideI,
                    float          *Y_re,
                    float          *Y_im,
                    csipl_stride    strideY,
                    csipl_length    n);
```

Parameters

- `X_re`, real part of complex vector, length m, input.
- `X_im`, imaginary part of complex vector, length m, input.
- `strideX`, integer scalar, input.
- `I_re`, real part of vector-index vector, length n, input.
- `I_im`, imaginary part of vector-index vector, length n, input.
- `strideI`, integer scalar, input.
- `Y_re`, real part of complex vector, length n, output.
- `Y_im`, imaginary part of complex vector, length n, output.
- `strideY`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`Y`$[j]$:= `X`$[$`I`$[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

csipl_mgather_P

The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void csipl_mgather_P(
                    scalar_P          *X,
                    int                ldX,
                    csipl_scalar_mi   *I,
                    csipl_stride       strideI,
                    scalar_P          *Y,
                    csipl_stride       strideY,
                    csipl_length       n);
```

The following instances are supported:

- `csipl_mgather_f`
- `csipl_mgather_i`
- `csipl_mgather_si`

Parameters

- `X`, matrix, size m by n, input.
- `ldX`, integer scalar, input.
- `I`, matrix-index vector, length p, input.
- `strideI`, integer scalar, input.
- `Y`, vector, length p, output.
- `strideY`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`Y`$[j]$:= `X`$[$`I`$[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

csipl_cmgather_inter_f

The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void csipl_cmgather_inter_f(
                    void              *X,
                    int                ldX,
                    csipl_scalar_mi   *I,
                    csipl_stride       strideI,
                    void              *Y,
                    csipl_stride       strideY,
                    csipl_length       n);
```

Parameters

- X, complex matrix, size m by n, input.
- ldX, integer scalar, input.
- I, matrix-index vector, length p, input.
- strideI, integer scalar, input.
- Y, complex vector, length p, output.
- strideY, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[j] := \mathtt{X}[\mathtt{I}[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

csipl_cmgather_split_f

The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void csipl_cmgather_split_f(
                    float            *X_re,
                    float            *X_im,
                    int               ldX,
                    csipl_scalar_mi  *I_re,
                    csipl_scalar_mi  *I_im,
                    csipl_stride      strideI,
                    float            *Y_re,
                    float            *Y_im,
                    csipl_stride      strideY,
                    csipl_length      n);
```

Parameters

- `X_re`, real part of complex matrix, size m by n, input.
- `X_im`, imaginary part of complex matrix, size m by n, input.
- `ldX`, integer scalar, input.
- `I_re`, real part of matrix-index vector, length p, input.
- `I_im`, imaginary part of matrix-index vector, length p, input.
- `strideI`, integer scalar, input.
- `Y_re`, real part of complex vector, length p, output.
- `Y_im`, imaginary part of complex vector, length p, output.
- `strideY`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[j] := \mathtt{X}[\mathtt{I}[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

csipl_vimag_inter_f

Extract the imaginary part of a complex vector.

Prototype

```
void csipl_vimag_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \text{imag}(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vimag_split_f

Extract the imaginary part of a complex vector.

Prototype

```
void csipl_vimag_split_f(
                    float         *A_re,
                    float         *A_im,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathrm{imag}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mimag_f

Extract the imaginary part of a complex matrix.

Prototype

```
void csipl_mimag_f(
                    void          *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \text{imag}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vpolar_inter_f

Convert a complex vector from rectangular to polar form. The polar data consists of a real vector containing the radius and a corresponding real vector containing the argument (angle) of the complex input data.

Prototype

```
void csipl_vpolar_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    float         *P,
                    csipl_stride   strideP,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- P, real vector, length n, output.
- strideP, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ and $\mathtt{P}[j * \mathtt{strideP}] := \arg(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vpolar_split_f

Convert a complex vector from rectangular to polar form. The polar data consists of a real vector containing the radius and a corresponding real vector containing the argument (angle) of the complex input data.

Prototype

```
void csipl_vpolar_split_f(
                    float         *A_re,
                    float         *A_im,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    float         *P,
                    csipl_stride   strideP,
                    csipl_length   n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- P, real vector, length n, output.
- strideP, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ and $\mathtt{P}[j * \mathtt{strideP}] := \arg(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mpolar_inter_f

Convert a complex matrix from rectangular to polar form. The polar data consists of a real matrix containing the radius and a corresponding real matrix containing the argument (angle) of the complex input data.

Prototype

```
void csipl_mpolar_inter_f(
                    void          *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    float         *P,
                    int            ldP,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- P, real matrix, size m by n, output.
- ldP, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|$ and $\mathtt{P}[j * \mathtt{ldP} + k] := \arg(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_mpolar_split_f

Convert a complex matrix from rectangular to polar form. The polar data consists of a real matrix containing the radius and a corresponding real matrix containing the argument (angle) of the complex input data.

Prototype

```
void csipl_mpolar_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R,
                    int           ldR,
                    float        *P,
                    int           ldP,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- P, real matrix, size m by n, output.
- ldP, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|$ and $\mathtt{P}[j * \mathtt{ldP} + k] := \arg(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vreal_inter_f

Extract the real part of a complex vector.

Prototype

```
void csipl_vreal_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    float         *R,
                    csipl_stride   strideR,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \text{real}(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_vreal_split_f

Extract the real part of a complex vector.

Prototype

```
void csipl_vreal_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *R,
                    csipl_stride  strideR,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- R, real vector, length n, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R$[j * \texttt{strideR}] := \text{real}(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

csipl_mreal_f

Extract the real part of a complex matrix.

Prototype

```
void csipl_mreal_f(
                    void          *A,
                    int            ldA,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathrm{real}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

csipl_vrect_inter_f

Convert a pair of real vectors from complex polar to complex rectangular form.

Prototype

```
void csipl_vrect_inter_f(
                    float        *R,
                    csipl_stride  strideR,
                    float        *P,
                    csipl_stride  strideP,
                    void         *A,
                    csipl_stride  strideA,
                    csipl_length  n);
```

Parameters

- R, real vector, length n, input.
- strideR, integer scalar, input.
- P, real vector, length n, input.
- strideP, integer scalar, input.
- A, complex vector, length n, output.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{R}[j * \mathtt{strideR}] \cdot (\cos(\mathtt{P}[j * \mathtt{strideP}]) + \mathrm{i} \cdot \sin(\mathtt{P}[j * \mathtt{strideP}]))$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real vectors.

csipl_vrect_split_f

Convert a pair of real vectors from complex polar to complex rectangular form.

Prototype

```
void csipl_vrect_split_f(
                    float         *R,
                    csipl_stride   strideR,
                    float         *P,
                    csipl_stride   strideP,
                    float         *A_re,
                    float         *A_im,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- R, real vector, length n, input.
- strideR, integer scalar, input.
- P, real vector, length n, input.
- strideP, integer scalar, input.
- A_re, real part of complex vector, length n, output.
- A_im, imaginary part of complex vector, length n, output.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{R}[j * \mathtt{strideR}] \cdot (\cos(\mathtt{P}[j * \mathtt{strideP}]) + \mathrm{i} \cdot \sin(\mathtt{P}[j * \mathtt{strideP}]))$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real vectors.

csipl_mrect_inter_f

Convert a pair of real matrices from complex polar to complex rectangular form.

Prototype

```
void csipl_mrect_inter_f(
                    float        *R,
                    int           ldR,
                    float        *P,
                    int           ldP,
                    void         *A,
                    int           ldA,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- R, real matrix, size m by n, input.
- ldR, integer scalar, input.
- P, real matrix, size m by n, input.
- ldP, integer scalar, input.
- A, complex matrix, size m by n, output.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{A}[j*\mathtt{ldA}+k] := \mathtt{R}[j*\mathtt{ldR}+k]\cdot(\cos(\mathtt{P}[j*\mathtt{ldP}+k])+\mathrm{i}\cdot\sin(\mathtt{P}[j*\mathtt{ldP}+k]))$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real matrices.

csipl_mrect_split_f

Convert a pair of real matrices from complex polar to complex rectangular form.

Prototype

```
void csipl_mrect_split_f(
                    float        *R,
                    int           ldR,
                    float        *P,
                    int           ldP,
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- R, real matrix, size m by n, input.
- ldR, integer scalar, input.
- P, real matrix, size m by n, input.
- ldP, integer scalar, input.
- A_re, real part of complex matrix, size m by n, output.
- A_im, imaginary part of complex matrix, size m by n, output.
- ldA, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{A}[j*\mathtt{ldA}+k] := \mathtt{R}[j*\mathtt{ldR}+k]\cdot(\cos(\mathtt{P}[j*\mathtt{ldP}+k])+\mathrm{i}\cdot\sin(\mathtt{P}[j*\mathtt{ldP}+k]))$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real matrices.

csipl_vscatter_P

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.

Prototype

```
void csipl_vscatter_P(
                    scalar_P       *X,
                    csipl_stride    strideX,
                    scalar_P       *Y,
                    csipl_stride    strideY,
                    unsigned int   *I,
                    csipl_stride    strideI,
                    csipl_length    n);
```

The following instances are supported:

```
csipl_vscatter_f
csipl_vscatter_i
csipl_vscatter_si
```

Parameters

- X, vector, length n, input.
- strideX, integer scalar, input.
- Y, vector, length m, output.
- strideY, integer scalar, input.
- I, vector-index vector, length n, input.
- strideI, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[\mathtt{I}[j]] := \mathtt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

csipl_cvscatter_inter_f

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.

Prototype

```
void csipl_cvscatter_inter_f(
                    void          *X,
                    csipl_stride   strideX,
                    void          *Y,
                    csipl_stride   strideY,
                    unsigned int  *I,
                    csipl_stride   strideI,
                    csipl_length   n);
```

Parameters

- X, complex vector, length n, input.
- strideX, integer scalar, input.
- Y, complex vector, length m, output.
- strideY, integer scalar, input.
- I, vector-index vector, length n, input.
- strideI, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[\mathtt{I}[j]] := \mathtt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

csipl_cvscatter_split_f

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.

Prototype

```
void csipl_cvscatter_split_f(
                    float          *X_re,
                    float          *X_im,
                    csipl_stride    strideX,
                    float          *Y_re,
                    float          *Y_im,
                    csipl_stride    strideY,
                    unsigned int   *I_re,
                    unsigned int   *I_im,
                    csipl_stride    strideI,
                    csipl_length    n);
```

Parameters

- `X_re`, real part of complex vector, length n, input.
- `X_im`, imaginary part of complex vector, length n, input.
- `strideX`, integer scalar, input.
- `Y_re`, real part of complex vector, length m, output.
- `Y_im`, imaginary part of complex vector, length m, output.
- `strideY`, integer scalar, input.
- `I_re`, real part of vector-index vector, length n, input.
- `I_im`, imaginary part of vector-index vector, length n, input.
- `strideI`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[\mathtt{I}[j]] := \mathtt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

csipl_mscatter_P

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.

Prototype

```
void csipl_mscatter_P(
                    scalar_P          *X,
                    csipl_stride       strideX,
                    scalar_P          *Y,
                    int                ldY,
                    csipl_scalar_mi   *I,
                    csipl_stride       strideI,
                    csipl_length       n);
```

The following instances are supported:

- `csipl_mscatter_f`
- `csipl_mscatter_i`
- `csipl_mscatter_si`

Parameters

- `X`, vector, length p, input.
- `strideX`, integer scalar, input.
- `Y`, matrix, size m by n, output.
- `ldY`, integer scalar, input.
- `I`, matrix-index vector, length p, input.
- `strideI`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\texttt{Y}[\texttt{I}[j]] := \texttt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

csipl_cmscatter_inter_f

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.

Prototype

```
void csipl_cmscatter_inter_f(
                    void              *X,
                    csipl_stride       strideX,
                    void              *Y,
                    int                ldY,
                    csipl_scalar_mi   *I,
                    csipl_stride       strideI,
                    csipl_length       n);
```

Parameters

- X, complex vector, length p, input.
- strideX, integer scalar, input.
- Y, complex matrix, size m by n, output.
- ldY, integer scalar, input.
- I, matrix-index vector, length p, input.
- strideI, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[\mathtt{I}[j]] := \mathtt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

csipl_cmscatter_split_f

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.

Prototype

```
void csipl_cmscatter_split_f(
                    float             *X_re,
                    float             *X_im,
                    csipl_stride       strideX,
                    float             *Y_re,
                    float             *Y_im,
                    int                ldY,
                    csipl_scalar_mi   *I_re,
                    csipl_scalar_mi   *I_im,
                    csipl_stride       strideI,
                    csipl_length       n);
```

Parameters

- `X_re`, real part of complex vector, length p, input.
- `X_im`, imaginary part of complex vector, length p, input.
- `strideX`, integer scalar, input.
- `Y_re`, real part of complex matrix, size m by n, output.
- `Y_im`, imaginary part of complex matrix, size m by n, output.
- `ldY`, integer scalar, input.
- `I_re`, real part of matrix-index vector, length p, input.
- `I_im`, imaginary part of matrix-index vector, length p, input.
- `strideI`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y}[\mathtt{I}[j]] := \mathtt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

csipl_vswap_P

Swap elements between two vectors.

Prototype

```
void csipl_vswap_P(
                    scalar_P      *A,
                    csipl_stride   strideA,
                    scalar_P      *B,
                    csipl_stride   strideB,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vswap_f
- csipl_vswap_i
- csipl_vswap_si

Parameters

- A, vector, length n, modified in place.
- strideA, integer scalar, input.
- B, vector, length n, modified in place.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$t := \mathtt{A}[j * \mathtt{strideA}]$
$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{B}[j * \mathtt{strideB}]$
$\mathtt{B}[j * \mathtt{strideB}] := t$
where $0 \leqslant j < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

csipl_cvswap_inter_f

Swap elements between two vectors.

Prototype

```
void csipl_cvswap_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, modified in place.
- strideA, integer scalar, input.
- B, complex vector, length n, modified in place.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$t := \mathtt{A}[j * \mathtt{strideA}]$
$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{B}[j * \mathtt{strideB}]$
$\mathtt{B}[j * \mathtt{strideB}] := t$
where $0 \leqslant j < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

csipl_cvswap_split_f

Swap elements between two vectors.

Prototype

```
void csipl_cvswap_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, modified in place.
- A_im, imaginary part of complex vector, length n, modified in place.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, modified in place.
- B_im, imaginary part of complex vector, length n, modified in place.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$t := \mathtt{A}[j * \mathtt{strideA}]$
$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{B}[j * \mathtt{strideB}]$
$\mathtt{B}[j * \mathtt{strideB}] := t$
where $0 \leqslant j < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

csipl_mswap_P

Swap elements between two matrices.

Prototype

```
void csipl_mswap_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_mswap_f
csipl_mswap_i
csipl_mswap_si
```

Parameters

- A, matrix, size m by n, modified in place.
- ldA, integer scalar, input.
- B, matrix, size m by n, modified in place.
- ldB, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$t := \mathtt{A}[j * \mathtt{ldA} + k]$
$\mathtt{A}[j * \mathtt{ldA} + k] := \mathtt{B}[j * \mathtt{ldB} + k]$
$\mathtt{B}[j * \mathtt{ldB} + k] := t$
where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

csipl_cmswap_inter_f

Swap elements between two matrices.

Prototype

```
void csipl_cmswap_inter_f(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, modified in place.
- ldA, integer scalar, input.
- B, complex matrix, size m by n, modified in place.
- ldB, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$t := \mathtt{A}[j * \mathtt{ldA} + k]$
$\mathtt{A}[j * \mathtt{ldA} + k] := \mathtt{B}[j * \mathtt{ldB} + k]$
$\mathtt{B}[j * \mathtt{ldB} + k] := t$
where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

csipl_cmswap_split_f

Swap elements between two matrices.

Prototype

```
void csipl_cmswap_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *B_re,
                    float        *B_im,
                    int           ldB,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, modified in place.
- A_im, imaginary part of complex matrix, size m by n, modified in place.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size m by n, modified in place.
- B_im, imaginary part of complex matrix, size m by n, modified in place.
- ldB, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$t := \mathtt{A}[j * \mathtt{ldA} + k]$
$\mathtt{A}[j * \mathtt{ldA} + k] := \mathtt{B}[j * \mathtt{ldB} + k]$
$\mathtt{B}[j * \mathtt{ldB} + k] := t$
where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

Chapter 7. Signal Processing Functions

7.1 FFT Functions

- csipl_ccfftip_create_f
- csipl_ccfftop_create_f
- csipl_crfftop_create_f
- csipl_rcfftop_create_f
- csipl_ccfftip_inter_f
- csipl_ccfftip_split_f
- csipl_ccfftop_inter_f
- csipl_ccfftop_split_f
- csipl_crfftop_inter_f
- csipl_crfftop_split_f
- csipl_rcfftop_inter_f
- csipl_rcfftop_split_f
- csipl_fft_destroy_f
- csipl_fft_getattr_f
- csipl_ccfftmop_create_f
- csipl_ccfftmip_create_f
- csipl_crfftmop_create_f
- csipl_rcfftmop_create_f
- csipl_fftm_destroy_f
- csipl_fftm_getattr_f
- csipl_ccfftmip_inter_f
- csipl_ccfftmip_split_f
- csipl_ccfftmop_inter_f
- csipl_ccfftmop_split_f
- csipl_crfftmop_inter_f
- csipl_crfftmop_split_f
- csipl_rcfftmop_inter_f
- csipl_rcfftmop_split_f
- csipl_ccfft2dop_create_f
- csipl_ccfft2dip_create_f

- csipl_crfft2dop_create_f
- csipl_rcfft2dop_create_f
- csipl_ccfft2dip_inter_f
- csipl_ccfft2dip_split_f
- csipl_ccfft2dop_inter_f
- csipl_ccfft2dop_split_f
- csipl_crfft2dop_inter_f
- csipl_crfft2dop_split_f
- csipl_rcfft2dop_inter_f
- csipl_rcfft2dop_split_f
- csipl_fft2d_destroy_f
- csipl_fft2d_getattr_f

csipl_ccfftip_create_f

Create a 1D FFT object.

Prototype

```
csipl_fft_f * csipl_ccfftip_create_f(
                    csipl_index    length,
                    float          scale,
                    csipl_fft_dir  dir,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `length`, vector-index scalar, input. The length N of the data vector.
- `scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.
- `dir`, enumerated type, input.

`CSIPL_FFT_FWD`	forward
`CSIPL_FFT_INV`	reverse (or inverse)

- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: complex-to-complex in-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive and non-zero.
2. `dir` must be valid.
3. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

csipl_ccfftop_create_f

Create a 1D FFT object.

Prototype

```
csipl_fft_f * csipl_ccfftop_create_f(
                    csipl_index    length,
                    float          scale,
                    csipl_fft_dir  dir,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `length`, vector-index scalar, input. The length N of the data vector.
- `scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.
- `dir`, enumerated type, input.

`CSIPL_FFT_FWD`	forward
`CSIPL_FFT_INV`	reverse (or inverse)

- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: complex-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive and non-zero.
2. `dir` must be valid.
3. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

csipl_crfftop_create_f

Create a 1D FFT object.

Prototype

```
csipl_fft_f * csipl_crfftop_create_f(
                    csipl_index    length,
                    float          scale,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `length`, vector-index scalar, input. The length N of the data vector.
- `scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.
- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: (reverse) complex-to-real out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.

`NULL` is returned if the create fails.

Restrictions

The length N must be even.

Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

csipl_rcfftop_create_f

Create a 1D FFT object.

Prototype

```
csipl_fft_f * csipl_rcfftop_create_f(
                    csipl_index    length,
                    float          scale,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `length`, vector-index scalar, input. The length N of the data vector.
- `scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.
- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: (forward) real-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.

`NULL` is returned if the create fails.

Restrictions

The length N must be even.

Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

csipl_ccfftip_inter_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftip_inter_f(
                    csipl_fft_f   *plan,
                    void          *xy,
                    csipl_stride   stridexy);
```

Parameters

- plan, structure, input.
- xy, complex vector, modified in place.
- stridexy, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex in-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place FFT object.
3. The input must be a complex vector of length N, conformant to the FFT object.

Notes

csipl_ccfftip_split_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftip_split_f(
                    csipl_fft_f   *plan,
                    float         *xy_re,
                    float         *xy_im,
                    csipl_stride   stridexy);
```

Parameters

- plan, structure, input.
- xy_re, real part of complex vector, modified in place.
- xy_im, imaginary part of complex vector, modified in place.
- stridexy, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex in-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place FFT object.
3. The input must be a complex vector of length N, conformant to the FFT object.

Notes

csipl_ccfftop_inter_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftop_inter_f(
                    csipl_fft_f   *plan,
                    void          *x,
                    csipl_stride   stridex,
                    void          *y,
                    csipl_stride   stridey);
```

Parameters

- plan, structure, input.
- x, complex vector, input.
- stridex, integer scalar, input.
- y, complex vector, output.
- stridey, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex out-of-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place FFT object.

3. The input and output must be complex vectors of length N, conformant to the FFT object.

4. The input and output vectors must not overlap.

Notes

csipl_ccfftop_split_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftop_split_f(
                    csipl_fft_f   *plan,
                    float         *x_re,
                    float         *x_im,
                    csipl_stride   stridex,
                    float         *y_re,
                    float         *y_im,
                    csipl_stride   stridey);
```

Parameters

- plan, structure, input.
- x_re, real part of complex vector, input.
- x_im, imaginary part of complex vector, input.
- stridex, integer scalar, input.
- y_re, real part of complex vector, output.
- y_im, imaginary part of complex vector, output.
- stridey, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex out-of-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place FFT object.
3. The input and output must be complex vectors of length N, conformant to the FFT object.
4. The input and output vectors must not overlap.

Notes

csipl_crfftop_inter_f

Apply a complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void csipl_crfftop_inter_f(
                    csipl_fft_f   *plan,
                    void          *x,
                    csipl_stride   stridex,
                    float         *y,
                    csipl_stride   stridey);
```

Parameters

- plan, structure, input.
- x, complex vector, input.
- stridex, integer scalar, input.
- y, real vector, output.
- stridey, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-real out-of-place (reverse) Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the real vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(+2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride vectors are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place FFT object.

3. The input must be a complex vector of length $N/2 + 1$, conformant to the FFT object.
4. The output must be a real vector of even length N, conformant to the FFT object.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work.

For the output sequence, y, to be a real sequence, the following identity on the input sequence, x, must be true: $x_j = x^*_{N-j}$ for $\lfloor N/2 \rfloor < j < N$.

The input values x_j for $j > \lfloor N/2 \rfloor$ need not be supplied; they can be inferred from the first half of the input.

Thus, in the complex-to-real routine, x is a complex vector of length $\lfloor N/2 \rfloor + 1$ and y is a real vector of length N. Even though only $\lfloor N/2 \rfloor + 1$ input complex values are supplied, the size of the transform is still N in this case, because implicitly you are using the FFT formula for a sequence of length N.

The first value of the input vector, $x[0]$ must be a real number (that is, it must have zero imaginary part). The first value corresponds to the zero (DC) frequency component of the data. Since we restrict N to be an even number, the last value of the input vector, $x[N/2]$, must also be real. The last value corresponds to one half the Nyquist rate (or sample rate). This value is sometimes called the folding frequency. The routine assumes that these values are real; if you specify a non-zero imaginary part, it is ignored.

csipl_crfftop_split_f

Apply a complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void csipl_crfftop_split_f(
                    csipl_fft_f   *plan,
                    float         *x_re,
                    float         *x_im,
                    csipl_stride   stridex,
                    float         *y,
                    csipl_stride   stridey);
```

Parameters

- plan, structure, input.
- x_re, real part of complex vector, input.
- x_im, imaginary part of complex vector, input.
- stridex, integer scalar, input.
- y, real vector, output.
- stridey, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-real out-of-place (reverse) Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the real vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(+2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride vectors are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place FFT object.
3. The input must be a complex vector of length $N/2 + 1$, conformant to the FFT object.
4. The output must be a real vector of even length N, conformant to the FFT object.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work.

For the output sequence, y, to be a real sequence, the following identity on the input sequence, x, must be true: $x_j = x^*_{N-j}$ for $\lfloor N/2 \rfloor < j < N$.

The input values x_j for $j > \lfloor N/2 \rfloor$ need not be supplied; they can be inferred from the first half of the input.

Thus, in the complex-to-real routine, x is a complex vector of length $\lfloor N/2 \rfloor + 1$ and y is a real vector of length N. Even though only $\lfloor N/2 \rfloor + 1$ input complex values are supplied, the size of the transform is still N in this case, because implicitly you are using the FFT formula for a sequence of length N.

The first value of the input vector, $x[0]$ must be a real number (that is, it must have zero imaginary part). The first value corresponds to the zero (DC) frequency component of the data. Since we restrict N to be an even number, the last value of the input vector, $x[N/2]$, must also be real. The last value corresponds to one half the Nyquist rate (or sample rate). This value is sometimes called the folding frequency. The routine assumes that these values are real; if you specify a non-zero imaginary part, it is ignored.

csipl_rcfftop_inter_f

Apply a real-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_rcfftop_inter_f(
                    csipl_fft_f   *plan,
                    float         *x,
                    csipl_stride   stridex,
                    void          *y,
                    csipl_stride   stridey);
```

Parameters

- plan, structure, input.
- x, real vector, input.
- stridex, integer scalar, input.
- y, complex vector, output.
- stridey, integer scalar, input.

Return Value

- none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real vector x, and stores the results in the complex vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(-2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride views are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place FFT object.

3. The input must be a real vector of even length N.
4. The output must be a complex vector of length $N/2 + 1$.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.

The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. The theory of Fourier transforms tells us that for real input data, you have to compute only the first $\lfloor N/2 \rfloor + 1$ complex output values because the remaining values can be computed from the first half by the simple formula: $y_k = y^*_{N-k}$ for $\lfloor N/2 \rfloor < k < N$.

For real input data, the first output value, $y[0]$, will always be a real number (the imaginary part will be zero). The first output value is sometimes called the DC component of the FFT and corresponds to zero frequency. Since we restrict N to be an even number, $y[N/2]$ will also be real and thus, have zero imaginary part. The last value is called the folding frequency and is equal to one half the sample rate of the input data.

Thus, in the real-to-complex routine, x is a real array of even length N and y is a complex array of length $N/2 + 1$.

csipl_rcfftop_split_f

Apply a real-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_rcfftop_split_f(
                    csipl_fft_f   *plan,
                    float         *x,
                    csipl_stride   stridex,
                    float         *y_re,
                    float         *y_im,
                    csipl_stride   stridey);
```

Parameters

- plan, structure, input.
- x, real vector, input.
- stridex, integer scalar, input.
- y_re, real part of complex vector, output.
- y_im, imaginary part of complex vector, output.
- stridey, integer scalar, input.

Return Value

- none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real vector x, and stores the results in the complex vector y.

$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(-2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride views are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place FFT object.
3. The input must be a real vector of even length N.
4. The output must be a complex vector of length $N/2 + 1$.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.

The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. The theory of Fourier transforms tells us that for real input data, you have to compute only the first $\lfloor N/2 \rfloor + 1$ complex output values because the remaining values can be computed from the first half by the simple formula: $y_k = y^*_{N-k}$ for $\lfloor N/2 \rfloor < k < N$.

For real input data, the first output value, $y[0]$, will always be a real number (the imaginary part will be zero). The first output value is sometimes called the DC component of the FFT and corresponds to zero frequency. Since we restrict N to be an even number, $y[N/2]$ will also be real and thus, have zero imaginary part. The last value is called the folding frequency and is equal to one half the sample rate of the input data.

Thus, in the real-to-complex routine, x is a real array of even length N and y is a complex array of length $N/2 + 1$.

csipl_fft_destroy_f

Destroy an FFT object.

Prototype

```
int csipl_fft_destroy_f(
                    csipl_fft_f *plan);
```

Parameters

- plan, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) an FFT object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The input object must conform to the following:

1. The FFT object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

csipl_fft_getattr_f

Return the attributes of an FFT object.

Prototype

```
void csipl_fft_getattr_f(
                    csipl_fft_f       *plan,
                    csipl_fft_attr_f  *attr);
```

Parameters

- `plan`, structure, input.
- `attr`, pointer to structure, output.
  The attribute structure contains the following information:

`unsigned int`	`input`	input length
`unsigned int`	`output`	output length
`csipl_fft_place`	`place`	in-place or out-of-place
`float`	`scale`	scale factor
`csipl_fft_dir`	`dir`	forward or reverse

Return Value

- none.

Description

Returns the attributes of an FFT object.

Restrictions

Errors

The arguments must conform to the following:

1. The FFT object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

There is no attribute that explicitly indicates complex-to-complex, real-to-complex, or complex-to-real FFTs. This may be inferred by examining the input and output sizes.

csipl_ccfftmop_create_f

Create a 1D multiple FFT object.

Prototype

```
csipl_fftm_f * csipl_ccfftmop_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    csipl_fft_dir  dir,
                    csipl_major    major,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).
- `cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.
- `dir`, enumerated type, input.

  `CSIPL_FFT_FWD` forward
  `CSIPL_FFT_INV` reverse (or inverse)
- `major`, enumerated type, input.

  `CSIPL_ROW` apply operation to the rows
  `CSIPL_COL` apply operation to the columns
- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

  `CSIPL_ALG_SPACE` minimise memory usage
  `CSIPL_ALG_TIME` minimise execution time
  `CSIPL_ALG_NOISE` maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: complex-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `dir` must be valid.
3. `major` must be valid.
4. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.

FFT operations are supported for all values of N and M.

csipl_ccfftmip_create_f

Create a 1D multiple FFT object.

Prototype

```
csipl_fftm_f * csipl_ccfftmip_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    csipl_fft_dir  dir,
                    csipl_major    major,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).
- `cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.
- `dir`, enumerated type, input.

  `CSIPL_FFT_FWD` forward
  `CSIPL_FFT_INV` reverse (or inverse)
- `major`, enumerated type, input.

  `CSIPL_ROW` apply operation to the rows
  `CSIPL_COL` apply operation to the columns
- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

  `CSIPL_ALG_SPACE` minimise memory usage
  `CSIPL_ALG_TIME` minimise execution time
  `CSIPL_ALG_NOISE` maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: complex-to-complex in-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `dir` must be valid.
3. `major` must be valid.
4. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.

FFT operations are supported for all values of N and M.

csipl_crfftmop_create_f

Create a 1D multiple FFT object.

Prototype

```
csipl_fftm_f * csipl_crfftmop_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    csipl_major    major,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).
- `cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.
- `major`, enumerated type, input.

`CSIPL_ROW`	apply operation to the rows
`CSIPL_COL`	apply operation to the columns

- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: (reverse) complex-to-real out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.

`NULL` is returned if the create fails.

Restrictions

In the FFT direction, the length must be even and the stride must be 1.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `major` must be valid.
3. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.

FFT operations are supported for all values of N and M.

csipl_rcfftmop_create_f

Create a 1D multiple FFT object.

Prototype

```
csipl_fftm_f * csipl_rcfftmop_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    csipl_major    major,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).
- `cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.
- `major`, enumerated type, input.

`CSIPL_ROW`	apply operation to the rows
`CSIPL_COL`	apply operation to the columns

- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: (forward) real-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.

A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.

`NULL` is returned if the create fails.

Restrictions

In the FFT direction, the length must be even and the stride must be 1.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `major` must be valid.
3. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.

FFT operations are supported for all values of N and M.

csipl_fftm_destroy_f

Destroy an FFT object.

Prototype

```
int csipl_fftm_destroy_f(
                    csipl_fftm_f *plan);
```

Parameters

- plan, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) an FFT object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The input object must conform to the following:

1. The FFT object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

csipl_fftm_getattr_f

Return the attributes of an FFT object.

Prototype

```
void csipl_fftm_getattr_f(
                    csipl_fftm_f       *plan,
                    csipl_fftm_attr_f  *attr);
```

Parameters

- plan, structure, input.
- attr, pointer to structure, output.
  The attribute structure contains the following information:

csipl_scalar_mi	input	numbers of rows and columns in input matrix
csipl_scalar_mi	output	numbers of rows and columns in output matrix
csipl_fft_place	place	in-place or out-of-place
float	scale	scale factor
csipl_fft_dir	dir	forward or reverse
csipl_major	major	by row or column

Return Value

- none.

Description

Returns the attributes of an FFT object.

Restrictions

Errors

The arguments must conform to the following:

1. The FFT object plan must be valid.
2. The attribute pointer attr must not be NULL.

Notes

There is no attribute that explicitly indicates complex-to-complex, real-to-complex, or complex-to-real FFTs. This may be inferred by examining the input and output sizes.

csipl_ccfftmip_inter_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftmip_inter_f(
                    csipl_fftm_f  *plan,
                    void          *XY,
                    int            ldXY);
```

Parameters

- plan, structure, input.
- XY, complex matrix, modified in place.
- ldXY, integer scalar, input.

Return Value

- none.

Description

Computes multiple complex-to-complex in-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.

A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.

See csipl_ccfftip_split_f for more details.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place multiple FFT object.
3. The input must be a complex matrix of size M by N, conformant to the FFT object.

Notes

csipl_ccfftmip_split_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftmip_split_f(
                    csipl_fftm_f  *plan,
                    float         *XY_re,
                    float         *XY_im,
                    int            ldXY);
```

Parameters

- plan, structure, input.
- XY_re, real part of complex matrix, modified in place.
- XY_im, imaginary part of complex matrix, modified in place.
- ldXY, integer scalar, input.

Return Value

- none.

Description

Computes multiple complex-to-complex in-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.

A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.

See csipl_ccfftip_split_f for more details.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place multiple FFT object.
3. The input must be a complex matrix of size M by N, conformant to the FFT object.

Notes

csipl_ccfftmop_inter_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftmop_inter_f(
                    csipl_fftm_f  *plan,
                    void          *X,
                    int            ldX,
                    void          *Y,
                    int            ldY);
```

Parameters

- plan, structure, input.
- X, complex matrix, input.
- ldX, integer scalar, input.
- Y, complex matrix, output.
- ldY, integer scalar, input.

Return Value

- none.

Description

Computes multiple complex-to-complex out-of-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.

A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.

See csipl_ccfftop_split_f for more details.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.

2. The FFT object must be a complex-to-complex out-of-place multiple FFT object.
3. The input and output must be complex matrices of size M by N, conformant to the FFT object.
4. The input and output matrices must not overlap.

Notes

csipl_ccfftmop_split_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfftmop_split_f(
                    csipl_fftm_f  *plan,
                    float         *X_re,
                    float         *X_im,
                    int            ldX,
                    float         *Y_re,
                    float         *Y_im,
                    int            ldY);
```

Parameters

- `plan`, structure, input.
- `X_re`, real part of complex matrix, input.
- `X_im`, imaginary part of complex matrix, input.
- `ldX`, integer scalar, input.
- `Y_re`, real part of complex matrix, output.
- `Y_im`, imaginary part of complex matrix, output.
- `ldY`, integer scalar, input.

Return Value

- none.

Description

Computes multiple complex-to-complex out-of-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.

A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.

See `csipl_ccfftop_split_f` for more details.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place multiple FFT object.
3. The input and output must be complex matrices of size M by N, conformant to the FFT object.
4. The input and output matrices must not overlap.

Notes

csipl_crfftmop_inter_f

Apply a multiple complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void csipl_crfftmop_inter_f(
                    csipl_fftm_f  *plan,
                    void          *X,
                    int            ldX,
                    float         *Y,
                    int            ldY);
```

Parameters

- `plan`, structure, input.
- `X`, complex matrix, input.
- `ldX`, integer scalar, input.
- `Y`, real matrix, output.
- `ldY`, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-real out-of-place reverse Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.

A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.

See `csipl_crfftop_inter_f` for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The output length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place multiple FFT object.
3. The input must be a complex matrix of size:
   - By rows: M by $N/2+1$, N even.
   - By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The output must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

csipl_crfftmop_split_f

Apply a multiple complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void csipl_crfftmop_split_f(
                    csipl_fftm_f  *plan,
                    float         *X_re,
                    float         *X_im,
                    int            ldX,
                    float         *Y,
                    int            ldY);
```

Parameters

- plan, structure, input.
- X_re, real part of complex matrix, input.
- X_im, imaginary part of complex matrix, input.
- ldX, integer scalar, input.
- Y, real matrix, output.
- ldY, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-real out-of-place reverse Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.

A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.

See csipl_crfftop_inter_f for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The output length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place multiple FFT object.
3. The input must be a complex matrix of size:
   - By rows: M by $N/2+1$, N even.
   - By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The output must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

csipl_rcfftmop_inter_f

Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT).

Prototype

```
void csipl_rcfftmop_inter_f(
                    csipl_fftm_f  *plan,
                    float         *X,
                    int            ldX,
                    void          *Y,
                    int            ldY);
```

Parameters

- plan, structure, input.
- X, real matrix, input.
- ldX, integer scalar, input.
- Y, complex matrix, output.
- ldY, integer scalar, input.

Return Value

- none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.

A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.

See csipl_rcfftop_inter_f for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The input length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place multiple FFT object.
3. The output must be a complex matrix of size:
   - By rows: M by $N/2+1$, N even.
   - By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The input must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

csipl_rcfftmop_split_f

Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT).

Prototype

```
void csipl_rcfftmop_split_f(
                    csipl_fftm_f  *plan,
                    float         *X,
                    int            ldX,
                    float         *Y_re,
                    float         *Y_im,
                    int            ldY);
```

Parameters

- `plan`, structure, input.
- `X`, real matrix, input.
- `ldX`, integer scalar, input.
- `Y_re`, real part of complex matrix, output.
- `Y_im`, imaginary part of complex matrix, output.
- `ldY`, integer scalar, input.

Return Value

- none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.

A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.

See `csipl_rcfftop_inter_f` for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The input length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place multiple FFT object.
3. The output must be a complex matrix of size:
   - By rows: M by $N/2+1$, N even.
   - By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The input must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

csipl_ccfft2dop_create_f

Create a 2D FFT object.

Prototype

```
csipl_fft2d_f * csipl_ccfft2dop_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    csipl_fft_dir  dir,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Number of rows (M).
- `cols`, vector-index scalar, input. Number of columns (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.
- `dir`, enumerated type, input.

`CSIPL_FFT_FWD`	forward
`CSIPL_FFT_INV`	reverse (or inverse)

- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: complex-to-complex out-of-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive and non-zero.
2. `dir` must be valid.
3. `hint` must be valid.

Notes

FFT operations are supported for all legal values of M and N.

csipl_ccfft2dip_create_f

Create a 2D FFT object.

Prototype

```
csipl_fft2d_f * csipl_ccfft2dip_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    csipl_fft_dir  dir,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Number of rows (M).
- `cols`, vector-index scalar, input. Number of columns (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.
- `dir`, enumerated type, input.

`CSIPL_FFT_FWD`	forward
`CSIPL_FFT_INV`	reverse (or inverse)

- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: complex-to-complex in-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive and non-zero.
2. `dir` must be valid.
3. `hint` must be valid.

Notes

FFT operations are supported for all legal values of M and N.

csipl_crfft2dop_create_f

Create a 2D FFT object.

Prototype

```
csipl_fft2d_f * csipl_crfft2dop_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Number of rows (M).
- `cols`, vector-index scalar, input. Number of columns (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.
- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: (reverse) complex-to-real out-of-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.

`NULL` is returned if the create fails.

Restrictions

The FFT is restricted to views with unit stride in either the row or column direction. The lengths M and N must be even.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported for all legal values of M and N.

csipl_rcfft2dop_create_f

Create a 2D FFT object.

Prototype

```
csipl_fft2d_f * csipl_rcfft2dop_create_f(
                    csipl_index    rows,
                    csipl_index    cols,
                    float          scale,
                    unsigned int   ntimes,
                    csipl_alg_hint hint);
```

Parameters

- `rows`, vector-index scalar, input. Number of rows (M).
- `cols`, vector-index scalar, input. Number of columns (N).
- `scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.
- `ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

Return Value

- structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: (forward) real-to-complex out-of-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.

`NULL` is returned if the create fails.

Restrictions

The FFT is restricted to views with unit stride in either the row or column direction. The lengths M and N must be even.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported for all legal values of M and N.

csipl_ccfft2dip_inter_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfft2dip_inter_f(
                    csipl_fft2d_f  *plan,
                    void           *XY,
                    int             ldXY);
```

Parameters

- `plan`, structure, input.
- `XY`, complex matrix, modified in place.
- `ldXY`, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex in-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place 2D FFT object.
3. The input/output `XY` must be a complex matrix of size M by N (conformant with the 2D FFT object).

Notes

csipl_ccfft2dip_split_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfft2dip_split_f(
                    csipl_fft2d_f  *plan,
                    float          *XY_re,
                    float          *XY_im,
                    int             ldXY);
```

Parameters

- `plan`, structure, input.
- `XY_re`, real part of complex matrix, modified in place.
- `XY_im`, imaginary part of complex matrix, modified in place.
- `ldXY`, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex in-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place 2D FFT object.
3. The input/output `XY` must be a complex matrix of size M by N (conformant with the 2D FFT object).

Notes

csipl_ccfft2dop_inter_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfft2dop_inter_f(
                    csipl_fft2d_f  *plan,
                    void           *X,
                    int             ldX,
                    void           *Y,
                    int             ldY);
```

Parameters

- plan, structure, input.
- X, complex matrix, input.
- ldX, integer scalar, input.
- Y, complex matrix, output.
- ldY, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex out-of-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place 2D FFT object.

3. The input `X` must be a complex matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.

Notes

csipl_ccfft2dop_split_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_ccfft2dop_split_f(
                    csipl_fft2d_f  *plan,
                    float          *X_re,
                    float          *X_im,
                    int             ldX,
                    float          *Y_re,
                    float          *Y_im,
                    int             ldY);
```

Parameters

- `plan`, structure, input.
- `X_re`, real part of complex matrix, input.
- `X_im`, imaginary part of complex matrix, input.
- `ldX`, integer scalar, input.
- `Y_re`, real part of complex matrix, output.
- `Y_im`, imaginary part of complex matrix, output.
- `ldY`, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-complex out-of-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place 2D FFT object.
3. The input `X` must be a complex matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.

Notes

csipl_crfft2dop_inter_f

Apply a complex-to-real 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_crfft2dop_inter_f(
                    csipl_fft2d_f  *plan,
                    void           *X,
                    int             ldX,
                    float          *Y,
                    int             ldY);
```

Parameters

- plan, structure, input.
- X, complex matrix, input.
- ldX, integer scalar, input.
- Y, real matrix, output.
- ldY, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-real out-of-place (reverse) 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(+2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix Y must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.

2. The FFT object must be a complex-to-real out-of-place 2D FFT object.
3. The input `X` must be a complex matrix of size (conformant with the 2D FFT object):
   - By rows: M by $N/2+1$ where N is even
   - By columns: $M/2+1$ by N where M is even.
4. The output `Y` must be a real matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.
6. The output matrix must be unit stride in the row or column direction.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work. See `csipl_crfftop_inter_f` for more details.

csipl_crfft2dop_split_f

Apply a complex-to-real 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_crfft2dop_split_f(
                    csipl_fft2d_f *plan,
                    float         *X_re,
                    float         *X_im,
                    int            ldX,
                    float         *Y,
                    int            ldY);
```

Parameters

- plan, structure, input.
- X_re, real part of complex matrix, input.
- X_im, imaginary part of complex matrix, input.
- ldX, integer scalar, input.
- Y, real matrix, output.
- ldY, integer scalar, input.

Return Value

- none.

Description

Computes a complex-to-real out-of-place (reverse) 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(+2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix Y must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place 2D FFT object.
3. The input `X` must be a complex matrix of size (conformant with the 2D FFT object):
   - By rows: M by $N/2+1$ where N is even
   - By columns: $M/2+1$ by N where M is even.
4. The output `Y` must be a real matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.
6. The output matrix must be unit stride in the row or column direction.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work. See `csipl_crfftop_inter_f` for more details.

csipl_rcfft2dop_inter_f

Apply a real-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_rcfft2dop_inter_f(
                    csipl_fft2d_f  *plan,
                    float          *X,
                    int             ldX,
                    void           *Y,
                    int             ldY);
```

Parameters

- plan, structure, input.
- X, real matrix, input.
- ldX, integer scalar, input.
- Y, complex matrix, output.
- ldY, integer scalar, input.

Return Value

- none.

Description

Computes a real-to-complex out-of-place (forward) 2D Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(-2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix X must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.

2. The FFT object must be a real-to-complex out-of-place 2D FFT object.
3. The input `X` must be a real matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size (conformant with the 2D FFT object):
   - By rows: M by $N/2+1$ where N is even
   - By columns: $M/2+1$ by N where M is even.
5. The input and output matrices must not overlap.
6. The input matrix must be unit stride in the row or column direction.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.

This routine computes a real-to-complex transform along the unit stride dimension, followed by a complex-to-complex transform in the other dimension. The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. See `csipl_rcfftop_inter_f` for more details.

csipl_rcfft2dop_split_f

Apply a real-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void csipl_rcfft2dop_split_f(
                    csipl_fft2d_f  *plan,
                    float          *X,
                    int             ldX,
                    float          *Y_re,
                    float          *Y_im,
                    int             ldY);
```

Parameters

- `plan`, structure, input.
- `X`, real matrix, input.
- `ldX`, integer scalar, input.
- `Y_re`, real part of complex matrix, output.
- `Y_im`, imaginary part of complex matrix, output.
- `ldY`, integer scalar, input.

Return Value

- none.

Description

Computes a real-to-complex out-of-place (forward) 2D Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.

$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(-2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix `X` must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place 2D FFT object.
3. The input `X` must be a real matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size (conformant with the 2D FFT object):
   - By rows: M by $N/2+1$ where N is even
   - By columns: $M/2+1$ by N where M is even.
5. The input and output matrices must not overlap.
6. The input matrix must be unit stride in the row or column direction.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.

This routine computes a real-to-complex transform along the unit stride dimension, followed by a complex-to-complex transform in the other dimension. The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. See `csipl_rcfftop_inter_f` for more details.

csipl_fft2d_destroy_f

Destroy an FFT object.

Prototype

```
int csipl_fft2d_destroy_f(
                    csipl_fft2d_f *plan);
```

Parameters

- plan, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) an FFT object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The input object must conform to the following:

1. The FFT object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

csipl_fft2d_getattr_f

Return the attributes of an FFT object.

Prototype

```
void csipl_fft2d_getattr_f(
                    csipl_fft2d_f       *plan,
                    csipl_fft2d_attr_f  *attr);
```

Parameters

- plan, structure, input.
- attr, pointer to structure, output.
  The attribute structure contains the following information:

csipl_scalar_mi	input	input length
csipl_scalar_mi	output	output length
csipl_fft_place	place	in-place or out-of-place
float	scale	scale factor
csipl_fft_dir	dir	forward or reverse

Return Value

- none.

Description

Returns the attributes of an FFT object.

Restrictions

Errors

The arguments must conform to the following:

1. The FFT object plan must be valid.
2. The attribute pointer attr must not be NULL.

Notes

There is no attribute that explicitly indicates complex-to-complex, real-to-complex, or complex-to-real FFTs. This may be inferred by examining the input and output sizes.

7.2 Convolution/Correlation Functions

- csipl_conv1d_create_f
- csipl_conv1d_destroy_f
- csipl_conv1d_getattr_f
- csipl_convolve1d_f
- csipl_conv2d_create_f
- csipl_conv2d_destroy_f
- csipl_conv2d_getattr_f
- csipl_convolve2d_f
- csipl_Dcorr1d_create_P
- csipl_Dcorr1d_destroy_P
- csipl_Dcorr1d_getattr_P
- csipl_correlate1d_f
- csipl_ccorrelate1d_inter_f
- csipl_ccorrelate1d_split_f
- csipl_Dcorr2d_create_P
- csipl_Dcorr2d_destroy_P
- csipl_Dcorr2d_getattr_P
- csipl_correlate2d_f
- csipl_ccorrelate2d_inter_f
- csipl_ccorrelate2d_split_f

csipl_conv1d_create_f

Create a decimated 1D convolution filter object.

Prototype

```
csipl_conv1d_f * csipl_conv1d_create_f(
                    float                  *kernel,
                    csipl_stride            stridekernel,
                    csipl_symmetry          symm,
                    unsigned int            N,
                    unsigned int            D,
                    csipl_support_region    support,
                    unsigned int            ntimes,
                    csipl_alg_hint          hint,
                    csipl_length            n);
```

Parameters

- `kernel`, real vector, input. Vector of non-redundant filter coefficients. There are M in the non-symmetric case and $\lceil M/2 \rceil$ for symmetric filters.
- `stridekernel`, integer scalar, input.
- `symm`, enumerated type, input.

`CSIPL_NONSYM`	non-symmetric
`CSIPL_SYM_EVEN_LEN_ODD`	(even) symmetric, odd length
`CSIPL_SYM_EVEN_LEN_EVEN`	(even) symmetric, even length

- `N`, integer scalar, input. Length of data vector.
- `D`, integer scalar, input. Decimation factor.
- `support`, enumerated type, input.

`CSIPL_SUPPORT_FULL`	maximum region
`CSIPL_SUPPORT_SAME`	input and output same size
`CSIPL_SUPPORT_MIN`	region without zero extending the kernel

- `ntimes`, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

- `n`, integer scalar, input.

Return Value

- structure.

Description

Creates a decimated convolution filter object and returns a pointer to the object. The user specifies the kernel (filter order, symmetry, and filter coefficients), the region of support, and the integral output decimation factor.

If the create fails, `NULL` is returned.

A 1D convolution object is used to compute the convolution of a real filter (kernel) vector h of length M with a real data vector x of length N and output decimation factor D to produce an output vector y.

Full:
length $= \lfloor (N+M-2)/D \rfloor + 1$

$$y[k] := \sum_{j=0}^{M-1} h[j]\,\langle x[kD-j]\rangle \qquad \text{for } 0 \leqslant k \leqslant \lfloor (N+M-2)/D \rfloor.$$

Same size:
length $= \lfloor (N-1)/D \rfloor + 1$

$$y[k] := \sum_{j=0}^{M-1} h[j]\langle x[kD+\lfloor M/2 \rfloor - j]\rangle \qquad \text{for } 0 \leqslant k \leqslant \lfloor (N-1)/D \rfloor.$$

Minimum (not zero padded):
length $= \lfloor (N-1)/D \rfloor - \lfloor (M-1)/D \rfloor + 1$

$$y[k] := \sum_{j=0}^{M-1} h[j]\langle x[kD+(M-1)-j]\rangle \qquad \text{for } 0 \leqslant k \leqslant \lfloor (N-1)/D \rfloor - \lfloor (M-1)/D \rfloor.$$

Where $\langle x[j]\rangle = \begin{cases} x[j] & : \quad 0 \leqslant j < N \\ 0 & : \quad \text{otherwise.} \end{cases}$

If the kernel is symmetric the redundant values are not specified.

Restrictions

The filter length must be less than or equal to the data length, $M \leqslant N$.

Errors

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments. The decimation factor, D, is normally one for non-lowpass filters.

csipl_conv1d_destroy_f

Destroy a 1D convolution object.

Prototype

```
int csipl_conv1d_destroy_f(
                    csipl_conv1d_f *plan);
```

Parameters

- plan, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) a 1D convolution object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The arguments must conform to the following:

1. The 1D convolution object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

csipl_conv1d_getattr_f

Returns the attributes for a 1D convolution object.

Prototype

```
void csipl_conv1d_getattr_f(
                    csipl_conv1d_f       *plan,
                    csipl_conv1d_attr_f  *attr);
```

Parameters

- `plan`, structure, input.
- `attr`, pointer to structure, output.
  The attribute structure contains the following information:

`unsigned int`	`kernel_len`	kernel length
`csipl_symmetry`	`symm`	kernel symmetry
`unsigned int`	`data_len`	data input length
`csipl_support_region`	`support`	output region of support
`unsigned int`	`out_len`	output length
`csipl_length`	`decimation`	output decimation factor

Return Value

- none.

Description

Returns the attributes for a 1D convolution object.

Restrictions

Errors

The arguments must conform to the following:

1. The 1D convolution object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

The length of the kernel is also known as the filter order.

csipl_convolve1d_f

Compute a decimated real one-dimensional (1D) convolution of two vectors.

Prototype

```
void csipl_convolve1d_f(
                    csipl_conv1d_f *plan,
                    float          *x,
                    csipl_stride    stridex,
                    float          *y,
                    csipl_stride    stridey,
                    csipl_length    n);
```

Parameters

- `plan`, structure, input.
- `x`, real vector, input.
- `stridex`, integer scalar, input.
- `y`, real vector, output.
- `stridey`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

Uses a 1D convolution object to compute the convolution of a real filter (kernel) vector h of length M with a real data vector x of length N and output decimation factor D to produce an output vector y.

See `csipl_conv1d_create_f` for details.

Restrictions

Errors

The arguments must conform to the following:

1. The 1D convolution object `plan` must be valid.

2. The input vector `x` must be of length N (conformant with the 1D convolution object).

3. The output vector `y` must be of length (conformant with the 1D convolution object):

   - Full: $\lfloor (N + M - 2)/D \rfloor + 1$
   - Same: $\lfloor (N - 1)/D \rfloor + 1$
   - Minimum: $\lfloor (N - 1)/D \rfloor - \lfloor (M - 1)/D \rfloor + 1$.

4. The input `x`, and the output `y`, must not overlap.

Notes

The decimation factor, D, is normally one for non-lowpass filters.

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments.

csipl_conv2d_create_f

Create a decimated 2D convolution filter object.

Prototype

```
csipl_conv2d_f * csipl_conv2d_create_f(
                    float                 *H,
                    int                    ldH,
                    csipl_symmetry         symm,
                    unsigned int           P,
                    unsigned int           Q,
                    unsigned int           D,
                    csipl_support_region   support,
                    unsigned int           ntimes,
                    csipl_alg_hint         hint,
                    csipl_length           m,
                    csipl_length           n);
```

Parameters

- H, real matrix, input. Matrix of non-redundant filter coefficients. There are M by N in the non-symmetric case and $\lceil M/2 \rceil$ by $\lceil N/2 \rceil$ for symmetric filters.
- ldH, integer scalar, input.
- symm, enumerated type, input.

CSIPL_NONSYM	non-symmetric
CSIPL_SYM_EVEN_LEN_ODD	(even) symmetric, odd length
CSIPL_SYM_EVEN_LEN_EVEN	(even) symmetric, even length

- P, integer scalar, input. Number of rows in data matrix.
- Q, integer scalar, input. Number of columns in data matrix.
- D, integer scalar, input. Decimation factor.
- support, enumerated type, input.

CSIPL_SUPPORT_FULL	maximum region
CSIPL_SUPPORT_SAME	input and output same size
CSIPL_SUPPORT_MIN	region without zero extending the kernel

- ntimes, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.
- hint, enumerated type, input.

CSIPL_ALG_SPACE	minimise memory usage
CSIPL_ALG_TIME	minimise execution time
CSIPL_ALG_NOISE	maximise numerical accuracy

- m, integer scalar, input.

- n, integer scalar, input.

Return Value

- structure.

Description

Creates a decimated 2D convolution filter object and returns a pointer to the object. The user specifies the kernel (filter order, symmetry, and filter coefficients), the region of support, and the integral output decimation factor.

If the create fails, NULL is returned.

A 2D convolution object is used to compute the convolution of a real filter (kernel) matrix H of size M by N with a real data matrix X of size P by Q, producing the output matrix Y. The filter size must be less than or equal to the size of the data.

Full:
size $= \lfloor (P+M-2)/D \rfloor + 1$ by $\lfloor (Q+N-2)/D \rfloor + 1$
$y[j,k] := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h[m,n] \langle x[jD-m, kD-n] \rangle$ for $0 \leqslant j \leqslant \lfloor (P+M-2)/D \rfloor$ and $0 \leqslant k \leqslant \lfloor (Q+N-2)/D \rfloor$.

Same size:
length $= \lfloor (P-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor + 1$
$y[j,k] := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h[m,n] \langle x[jD + \lfloor M/2 \rfloor - m, kD + \lfloor N/2 \rfloor - n] \rangle$ for $0 \leqslant j \leqslant \lfloor (P-1)/D \rfloor$ and $0 \leqslant k \leqslant \lfloor (Q-1)/D \rfloor$.

Minimum (not zero padded):
length $= \lfloor (P-1)/D \rfloor - \lfloor (M-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor - \lfloor (N-1)/D \rfloor + 1$.
$y[j,k] := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h[m,n] \langle x[jD + (M-1) - m, kD + (N-1) - n] \rangle$ for $0 \leqslant j \leqslant \lfloor (P-1)/D \rfloor - \lfloor (M-1)/D \rfloor$ and $0 \leqslant k \leqslant \lfloor (Q-1)/D \rfloor - \lfloor (N-1)/D \rfloor$.

Where $\langle x[j] \rangle = \{ \; x[j,k] \;\; : \;\; 0 \leqslant j < P$ and $0 \leqslant k < Q \;\; 0 \;\; : \;\; \text{otherwise}.$

The filter kernel can be even–symmetric or non-symmetric. If it is symmetric the redundant values are not specified.

Restrictions

The filter length must be less than or equal to the data length: $M \leqslant P$ and $N \leqslant Q$.

Memory major order must be the same for kernel, data and output.

The kernel, data, and output matrix must be unit stride in the major direction.

Errors

Notes

The symmetry, support and decimation attributes apply uniformly to all dimensions.

csipl_conv2d_destroy_f

Destroy a 2D convolution object.

Prototype

```
int csipl_conv2d_destroy_f(
                    csipl_conv2d_f *plan);
```

Parameters

- plan, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) a 2D convolution object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The arguments must conform to the following:

1. The 2D convolution object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

csipl_conv2d_getattr_f

Returns the attributes for a 2D convolution object.

Prototype

```
void csipl_conv2d_getattr_f(
                    csipl_conv2d_f       *plan,
                    csipl_conv2d_attr_f  *attr);
```

Parameters

- plan, structure, input.
- attr, pointer to structure, output.
  The attribute structure contains the following information:

csipl_scalar_mi	kernel_size	kernel size
csipl_symmetry	symm	kernel symmetry
csipl_scalar_mi	in_size	data input size
csipl_support_region	support	output region of support
csipl_scalar_mi	out_size	output size
csipl_length	decimation	output decimation factor

Return Value

- none.

Description

Returns the attributes for a 2D convolution object.

Restrictions

Errors

The arguments must conform to the following:

1. The 2D convolution object plan must be valid.
2. The attribute pointer attr must not be NULL.

Notes

The size of the kernel is also known as the filter order.

csipl_convolve2d_f

Compute a decimated real two-dimensional (2D) convolution of two matrices.

Prototype

```
void csipl_convolve2d_f(
                    csipl_conv2d_f  *plan,
                    float           *x,
                    int              ldx,
                    float           *y,
                    int              ldy,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- `plan`, structure, input.
- `x`, real matrix, size m by n, input.
- `ldx`, integer scalar, input.
- `y`, real matrix, size p by q, output.
- `ldy`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

Uses a 2D convolution object to compute the convolution of a real filter (kernel) matrix H of size M by N with a real data matrix X of size P by Q, producing the output matrix Y. The filter size must be less than or equal to the size of the data.

See `csipl_conv2d_create_f` for details.

Restrictions

Memory major order must be the same for kernel, data and output. The kernel, data, and output matrix are restricted to unit stride in the major direction.

Errors

The arguments must conform to the following:

1. The 2D convolution object `plan` must be valid.
2. The input matrix `x` must be of size P by Q (conformant with the 2D convolution object).
3. The output matrix `y` must be of size (conformant with the 2D convolution object):
   - Full: $\lfloor (P+M-2)/D \rfloor + 1$ by $\lfloor (Q+N-2)/D \rfloor + 1$
   - Same: $\lfloor (P-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor + 1$
   - Minimum: $\lfloor (P-1)/D \rfloor - \lfloor (M-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor - \lfloor (N-1)/D \rfloor + 1$.
4. The input `x`, and the output `y`, must not overlap.

Notes

The decimation factor, D, is normally one for non-lowpass filters.

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments.

csipl_Dcorr1d_create_P

Create a 1D correlation object.

Prototype

```
csipl_Dcorr1d_P * csipl_Dcorr1d_create_P(
                    unsigned int         M,
                    unsigned int         N,
                    csipl_support_region support,
                    unsigned int         ntimes,
                    csipl_alg_hint       hint);
```

The following instances are supported:

csipl_corr1d_create_f

csipl_ccorr1d_create_f

Parameters

- M, integer scalar, input. Length of input reference vector.
- N, integer scalar, input. Length of input data vector.
- support, enumerated type, input.

CSIPL_SUPPORT_FULL	maximum region
CSIPL_SUPPORT_SAME	input and output same size
CSIPL_SUPPORT_MIN	region without zero extending the kernel

  Output region of support (indicates which output points are computed).
- ntimes, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.
- hint, enumerated type, input.

CSIPL_ALG_SPACE	minimise memory usage
CSIPL_ALG_TIME	minimise execution time
CSIPL_ALG_NOISE	maximise numerical accuracy

Return Value

- structure.

Description

Creates a (cross-) correlation object and returns a pointer to the object. The user specifies the lengths of the reference vector r and the data vector x.

If the create fails, NULL is returned.

A 1D correlation object is used to compute the (cross-) correlation of a reference vector r of length M with a data vector x of length N to produce an output vector y.

Full:
length $= N + M - 1$

$$\widehat{y}[k] := \sum_{j=0}^{M-1} r[j]\,\langle x[k+j-M+1]^*\rangle \qquad \text{for } 0 \leqslant k \leqslant N+M-2$$

$$y[k] := \widehat{y}[k] * \begin{cases} 1/(k+1) & : \quad 0 \leqslant k < M-1 \\ 1/M & : \quad M-1 \leqslant k < N \\ 1/(N+M-1-k) & : \quad N \leqslant k < N+M-1. \end{cases}$$

Same size:
length $= N$

$$\widehat{y}[k] := \sum_{j=0}^{M-1} r[j]\langle x[k+j-\lfloor M/2\rfloor]^*\rangle \qquad \text{for } 0 \leqslant k \leqslant N-1$$

$$y[k] := \widehat{y}[k] * \begin{cases} 1/(k+\lceil M/2\rceil) & : \quad 0 \leqslant k < \lfloor M/2\rfloor \\ 1/M & : \quad \lfloor M/2\rfloor \leqslant k < N-\lfloor M/2\rfloor \\ 1/(N+\lfloor M/2\rfloor-k) & : \quad N-\lfloor M/2\rfloor \leqslant k < N. \end{cases}$$

Minimum (not zero padded):
length $= N - M + 1$

$$\widehat{y}[k] := \sum_{j=0}^{M-1} r[j]\langle x[k+j]^*\rangle \qquad \text{for } 0 \leqslant k \leqslant N-M$$

$$y[k] := \widehat{y}[k]/M.$$

Where $\langle x[j]\rangle = \begin{cases} x[j] & : \quad 0 \leqslant j < N \\ 0 & : \quad \text{otherwise.} \end{cases}$

The values $\widehat{y}[k]$ are the biased correlation estimates while the $y[k]$ are unbiased estimates. (The unbiased estimates are scaled by the number or terms in the summation for each lag where $\langle x[j]\rangle$ is not defined to be zero.)

Restrictions

The reference length must be less than or equal to the data length, $M \leqslant N$.

Errors

The arguments must conform to the following:

1. $1 \leqslant$ M $\leqslant$ N.
2. support must be valid.

3. `hint` must be valid.

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments: specify the FIR kernel as the reverse-indexed clone of the reference data.

csipl_Dcorr1d_destroy_P

Destroy a 1D correlation object.

Prototype

```
int csipl_Dcorr1d_destroy_P(
                    csipl_Dcorr1d_P *plan);
```

The following instances are supported:

```
csipl_corr1d_destroy_f
csipl_ccorr1d_destroy_f
```

Parameters

- `plan`, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) a 1D correlation object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The arguments must conform to the following:

1. The 1D correlation object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

csipl_Dcorr1d_getattr_P

Return the attributes for a 1D correlation object.

Prototype

```
void csipl_Dcorr1d_getattr_P(
                    csipl_Dcorr1d_P      *plan,
                    csipl_Dcorr1d_attr_P *attr);
```

The following instances are supported:

`csipl_corr1d_getattr_f`

`csipl_ccorr1d_getattr_f`

Parameters

- `plan`, structure, input.
- `attr`, pointer to structure, output.
  The attribute structure contains the following information:

`unsigned int`	`ref_len`	reference length
`unsigned int`	`data_len`	data input length
`csipl_support_region`	`support`	output region of support
`unsigned int`	`lag_len`	output (lags) length

Return Value

- none.

Description

Returns the attributes for a 1D correlation object.

Restrictions

Errors

The arguments must conform to the following:

1. The 1D correlation object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

csipl_correlate1d_f

Compute a real one-dimensional (1D) correlation of two vectors.

Prototype

```
void csipl_correlate1d_f(
                    csipl_corr1d_f *plan,
                    csipl_bias      bias,
                    float          *ref,
                    csipl_stride    strideref,
                    float          *x,
                    csipl_stride    stridex,
                    float          *y,
                    csipl_stride    stridey,
                    csipl_length    n);
```

Parameters

- plan, structure, input.
- bias, enumerated type, input.

CSIPL_BIASED	biased
CSIPL_UNBIASED	unbiased

  Return biased or unbiased estimates.
- ref, real vector, input.
- strideref, integer scalar, input.
- x, real vector, input.
- stridex, integer scalar, input.
- y, real vector, output.
- stridey, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Uses a 1D correlation object to compute the (cross-) correlation of a reference vector r of length M with a data vector x of length N to produce an output vector y.

See csipl_corr1d_create_f for details.

Restrictions

The reference length must be less than or equal to the data length, $M \leqslant N$.

Errors

The arguments must conform to the following:

1. The 1D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input vector `ref` must be of length M (conformant with the 1D correlation object).
4. The data input vector `x` must be of length N (conformant with the 1D correlation object).
5. The output vector `y` must be of length (conformant with the 1D correlation object):
   - Full: $N + M - 1$
   - Same: N
   - Minimum: $N - M + 1$.
6. The output `y` cannot overlap either of the input vector, `ref` or `x`.

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments: specify the FIR kernel as the reverse-indexed clone of the reference data.

csipl_ccorrelate1d_inter_f

Compute a real one-dimensional (1D) correlation of two vectors.

Prototype

```
void csipl_ccorrelate1d_inter_f(
                    csipl_ccorr1d_f *plan,
                    csipl_bias       bias,
                    void            *ref,
                    csipl_stride     strideref,
                    void            *x,
                    csipl_stride     stridex,
                    void            *y,
                    csipl_stride     stridey,
                    csipl_length     n);
```

Parameters

- plan, structure, input.
- bias, enumerated type, input.

CSIPL_BIASED	biased
CSIPL_UNBIASED	unbiased

  Return biased or unbiased estimates.
- ref, complex vector, input.
- strideref, integer scalar, input.
- x, complex vector, input.
- stridex, integer scalar, input.
- y, complex vector, output.
- stridey, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Uses a 1D correlation object to compute the (cross-) correlation of a reference vector r of length M with a data vector x of length N to produce an output vector y.

See csipl_ccorr1d_create_f for details.

Restrictions

The reference length must be less than or equal to the data length, $M \leqslant N$.

Errors

The arguments must conform to the following:

1. The 1D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input vector `ref` must be of length M (conformant with the 1D correlation object).
4. The data input vector `x` must be of length N (conformant with the 1D correlation object).
5. The output vector `y` must be of length (conformant with the 1D correlation object):
   - Full: $N + M - 1$
   - Same: N
   - Minimum: $N - M + 1$.
6. The output `y` cannot overlap either of the input vector, `ref` or `x`.

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments: specify the FIR kernel as the reverse-indexed clone of the reference data.

csipl_ccorrelate1d_split_f

Compute a real one-dimensional (1D) correlation of two vectors.

Prototype

```
void csipl_ccorrelate1d_split_f(
                    csipl_ccorr1d_f *plan,
                    csipl_bias       bias,
                    float           *ref_re,
                    float           *ref_im,
                    csipl_stride     strideref,
                    float           *x_re,
                    float           *x_im,
                    csipl_stride     stridex,
                    float           *y_re,
                    float           *y_im,
                    csipl_stride     stridey,
                    csipl_length     n);
```

Parameters

- `plan`, structure, input.
- `bias`, enumerated type, input.

`CSIPL_BIASED`	biased
`CSIPL_UNBIASED`	unbiased

  Return biased or unbiased estimates.
- `ref_re`, real part of complex vector, length m, input.
- `ref_im`, imaginary part of complex vector, length m, input.
- `strideref`, integer scalar, input.
- `x_re`, real part of complex vector, length n, input.
- `x_im`, imaginary part of complex vector, length n, input.
- `stridex`, integer scalar, input.
- `y_re`, real part of complex vector, length p, output.
- `y_im`, imaginary part of complex vector, length p, output.
- `stridey`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

Uses a 1D correlation object to compute the (cross-) correlation of a reference vector r of length M with a data vector x of length N to produce an output vector y.

See `csipl_ccorr1d_create_f` for details.

Restrictions

The reference length must be less than or equal to the data length, $M \leqslant N$.

Errors

The arguments must conform to the following:

1. The 1D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input vector `ref` must be of length M (conformant with the 1D correlation object).
4. The data input vector `x` must be of length N (conformant with the 1D correlation object).
5. The output vector `y` must be of length (conformant with the 1D correlation object):
   - Full: $N + M - 1$
   - Same: N
   - Minimum: $N - M + 1$.
6. The output `y` cannot overlap either of the input vector, `ref` or `x`.

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments: specify the FIR kernel as the reverse-indexed clone of the reference data.

csipl_Dcorr2d_create_P

Create a 2D correlation object.

Prototype

```
csipl_Dcorr2d_P * csipl_Dcorr2d_create_P(
                    unsigned int         M,
                    unsigned int         N,
                    unsigned int         P,
                    unsigned int         Q,
                    csipl_support_region support,
                    unsigned int         ntimes,
                    csipl_alg_hint       hint);
```

The following instances are supported:

csipl_corr2d_create_f

csipl_ccorr2d_create_f

Parameters

- M, integer scalar, input. Number of rows in reference matrix.
- N, integer scalar, input. Number of columns in reference matrix.
- P, integer scalar, input. Number of rows in data matrix.
- Q, integer scalar, input. Number of columns in data matrix.
- support, enumerated type, input.

CSIPL_SUPPORT_FULL	maximum region
CSIPL_SUPPORT_SAME	input and output same size
CSIPL_SUPPORT_MIN	region without zero extending the kernel

  Output region of support (indicates which output points are computed).
- ntimes, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.
- hint, enumerated type, input.

CSIPL_ALG_SPACE	minimise memory usage
CSIPL_ALG_TIME	minimise execution time
CSIPL_ALG_NOISE	maximise numerical accuracy

Return Value

- structure.

Description

Creates a (cross-) correlation object and returns a pointer to the object. The user specifies the sizes of the reference matrix R and the data matrix X.

If the create fails, `NULL` is returned.

A 2D correlation object is used to compute the (cross-) correlation of a reference matrix R of size M by N with a data matrix X of size P by Q to produce an output matrix Y.

Full:
size $= P + M - 1$ by $Q + N - 1$

$$\widehat{Y}[j,k] := \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} R[m,n]\,\langle X[m+j-M+1, n+k-N+1]^*\rangle \qquad \text{for } 0 \leqslant j \leqslant P+M-2$$

and $0 \leqslant k \leqslant Q + N - 2$

$$Y[j,k] := \widehat{Y}[j,k] * \begin{cases} 1/(j+1) & : \quad 0 \leqslant j < M-1 \\ 1/M & : \quad M-1 \leqslant j < P \\ 1/(P+M-1-i) & : \quad P \leqslant j < P+M-1; \\ 1/(k+1) & : \quad 0 \leqslant k < N-1 \\ 1/N & : \quad N-1 \leqslant k < Q \\ 1/(Q+N-1-k) & : \quad Q \leqslant k < Q+N-1. \end{cases}$$

Same size:
size $= P$ by Q

$$\widehat{Y}[j,k] := \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} R[j,k]\langle X[m+j-\lfloor M/2\rfloor, n+j-\lfloor N/2\rfloor]^*\rangle \qquad \text{for } 0 \leqslant j \leqslant P-1 \text{ and}$$

$0 \leqslant k \leqslant Q - 1$

$$Y[j,k] := \widehat{Y}[j,k] * \begin{cases} 1/(j+\lceil M/2\rceil) & : \quad 0 \leqslant j < \lfloor M/2\rfloor \\ 1/M & : \quad \lfloor M/2\rfloor \leqslant j < P-\lfloor M/2\rfloor \\ 1/(P+\lceil M/2\rceil-1-j) & : \quad P-\lfloor M/2\rfloor \leqslant j < P; \\ 1/(k+\lceil N/2\rceil) & : \quad 0 \leqslant k < \lfloor N/2\rfloor \\ 1/N & : \quad \lfloor N/2\rfloor \leqslant k < Q-\lfloor N/2\rfloor \\ 1/(Q+\lceil N/2\rceil-1-k) & : \quad Q-\lfloor N/2\rfloor \leqslant k < Q. \end{cases}$$

Minimum (not zero padded):
size $= P - M + 1$ by $Q - N + 1$

$$\widehat{Y}[j,k] := \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} R[j,k]\langle X[m+j, n+k]^*\rangle \qquad \text{for } 0 \leqslant j \leqslant P-M \text{ and } 0 \leqslant k \leqslant Q-N$$

$Y[k] := \widehat{Y}[j,k]/(M\,N)$.

Where $\langle x[j]\rangle = \begin{cases} x[j] & : \quad 0 \leqslant j < N \\ 0 & : \quad \text{otherwise.} \end{cases}$

The values $\widehat{Y}[j,k]$ are the biased correlation estimates while the $Y[j,k]$ are unbiased estimates. (The unbiased estimates are scaled by the number or terms in the summation for each lag where $\langle X[j,k]\rangle$ is not defined to be zero.)

Restrictions

The reference size must be less than or equal to the data size: $M \leqslant P$ and $N \leqslant Q$.

Errors

The arguments must conform to the following:

1. $1 \leqslant$ `M` $\leqslant$ `P` and $1 \leqslant$ `N` $\leqslant$ `Q`.
2. `support` must be valid.
3. `hint` must be valid.

Notes

The support attribute applies uniformly to all dimensions.

csipl_Dcorr2d_destroy_P

Destroy a 2D correlation object.

Prototype

```
int csipl_Dcorr2d_destroy_P(
                    csipl_Dcorr2d_P *plan);
```

The following instances are supported:

```
csipl_corr2d_destroy_f
csipl_ccorr2d_destroy_f
```

Parameters

- `plan`, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) a 2D correlation object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The arguments must conform to the following:

1. The 2D correlation object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

csipl_Dcorr2d_getattr_P

Return the attributes for a 2D correlation object.

Prototype

```
void csipl_Dcorr2d_getattr_P(
                    csipl_Dcorr2d_P      *plan,
                    csipl_Dcorr2d_attr_P *attr);
```

The following instances are supported:

csipl_corr2d_getattr_f

csipl_ccorr2d_getattr_f

Parameters

- plan, structure, input.
- attr, pointer to structure, output.
  The attribute structure contains the following information:

csipl_scalar_mi	ref_size	reference size
csipl_scalar_mi	data_size	data input size
csipl_support_region	support	output region of support
csipl_scalar_mi	out_size	output size

Return Value

- none.

Description

Returns the attributes for a 2D correlation object.

Restrictions

Errors

The arguments must conform to the following:

1. The 2D correlation object plan must be valid.
2. The attribute pointer attr must not be NULL.

Notes

csipl_correlate2d_f

Compute a two-dimensional (2D) correlation of two matrices.

Prototype

```
void csipl_correlate2d_f(
                    csipl_corr2d_f *plan,
                    csipl_bias      bias,
                    float          *ref,
                    int             ldref,
                    float          *x,
                    int             ldx,
                    float          *y,
                    int             ldy,
                    csipl_length    m,
                    csipl_length    n);
```

Parameters

- plan, structure, input.
- bias, enumerated type, input.

CSIPL_BIASED	biased
CSIPL_UNBIASED	unbiased

  Return biased or unbiased estimates.
- ref, real matrix, input.
- ldref, integer scalar, input.
- x, real matrix, input.
- ldx, integer scalar, input.
- y, real matrix, output.
- ldy, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Uses a 2D correlation object to compute the (cross-) correlation of a reference matrix R with a data matrix X to produce an output matrix Y.

See `csipl_corr2d_create_f` for details.

Restrictions

The reference size must be less than or equal to the data size: $M \leqslant P$ and $N \leqslant Q$. Memory major order must be the same for reference, data and output. The matrix views must be unit stride in the major direction.

Errors

The arguments must conform to the following:

1. The 2D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input matrix `ref` must be of size M by N (conformant with the 2D correlation object).
4. The data input matrix `x` must be of size P by Q (conformant with the 2D correlation object).
5. The output matrix `y` must be of size (conformant with the 2D correlation object):
   - Full: $P + M - 1$ by $Q + N - 1$
   - Same: P by Q
   - Minimum: $P - M + 1$ by $Q - N + 1$.
6. The output `y` cannot overlap either of the input matrices, `ref` or `x`.
7. Memory major order must be the same for reference, data and output.
8. The matrix views must be unit stride in the major direction.

Notes

csipl_ccorrelate2d_inter_f

Compute a two-dimensional (2D) correlation of two matrices.

Prototype

```
void csipl_ccorrelate2d_inter_f(
                    csipl_ccorr2d_f *plan,
                    csipl_bias       bias,
                    void            *ref,
                    int              ldref,
                    void            *x,
                    int              ldx,
                    void            *y,
                    int              ldy,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- plan, structure, input.
- bias, enumerated type, input.

CSIPL_BIASED	biased
CSIPL_UNBIASED	unbiased

  Return biased or unbiased estimates.
- ref, complex matrix, input.
- ldref, integer scalar, input.
- x, complex matrix, input.
- ldx, integer scalar, input.
- y, complex matrix, output.
- ldy, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Uses a 2D correlation object to compute the (cross-) correlation of a reference matrix R with a data matrix X to produce an output matrix Y.

See `csipl_ccorr2d_create_f` for details.

Restrictions

The reference size must be less than or equal to the data size: $M \leqslant P$ and $N \leqslant Q$. Memory major order must be the same for reference, data and output. The matrix views must be unit stride in the major direction.

Errors

The arguments must conform to the following:

1. The 2D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input matrix `ref` must be of size M by N (conformant with the 2D correlation object).
4. The data input matrix `x` must be of size P by Q (conformant with the 2D correlation object).
5. The output matrix `y` must be of size (conformant with the 2D correlation object):
   - Full: $P + M - 1$ by $Q + N - 1$
   - Same: P by Q
   - Minimum: $P - M + 1$ by $Q - N + 1$.
6. The output `y` cannot overlap either of the input matrices, `ref` or `x`.
7. Memory major order must be the same for reference, data and output.
8. The matrix views must be unit stride in the major direction.

Notes

csipl_ccorrelate2d_split_f

Compute a two-dimensional (2D) correlation of two matrices.

Prototype

```
void csipl_ccorrelate2d_split_f(
                    csipl_ccorr2d_f *plan,
                    csipl_bias       bias,
                    float           *ref_re,
                    float           *ref_im,
                    int              ldref,
                    float           *x_re,
                    float           *x_im,
                    int              ldx,
                    float           *y_re,
                    float           *y_im,
                    int              ldy,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- `plan`, structure, input.
- `bias`, enumerated type, input.

`CSIPL_BIASED`	biased
`CSIPL_UNBIASED`	unbiased

  Return biased or unbiased estimates.
- `ref_re`, real part of complex matrix, size m by n, input.
- `ref_im`, imaginary part of complex matrix, size m by n, input.
- `ldref`, integer scalar, input.
- `x_re`, real part of complex matrix, size p by q, input.
- `x_im`, imaginary part of complex matrix, size p by q, input.
- `ldx`, integer scalar, input.
- `y_re`, real part of complex matrix, size P by Q, output.
- `y_im`, imaginary part of complex matrix, size P by Q, output.
- `ldy`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

Uses a 2D correlation object to compute the (cross-) correlation of a reference matrix R with a data matrix X to produce an output matrix Y.

See `csipl_ccorr2d_create_f` for details.

Restrictions

The reference size must be less than or equal to the data size: $M \leqslant P$ and $N \leqslant Q$. Memory major order must be the same for reference, data and output. The matrix views must be unit stride in the major direction.

Errors

The arguments must conform to the following:

1. The 2D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input matrix `ref` must be of size M by N (conformant with the 2D correlation object).
4. The data input matrix `x` must be of size P by Q (conformant with the 2D correlation object).
5. The output matrix `y` must be of size (conformant with the 2D correlation object):
   - Full: $P + M - 1$ by $Q + N - 1$
   - Same: P by Q
   - Minimum: $P - M + 1$ by $Q - N + 1$.
6. The output `y` cannot overlap either of the input matrices, `ref` or `x`.
7. Memory major order must be the same for reference, data and output.
8. The matrix views must be unit stride in the major direction.

Notes

7.3 Window Functions

- csipl_vcreate_blackman_f
- csipl_vcreate_cheby_f
- csipl_vcreate_hanning_f
- csipl_vcreate_kaiser_f

csipl_vcreate_blackman_f

Create a vector with Blackman window weights.

Prototype

```
float  * csipl_vcreate_blackman_f(
                    unsigned int      N,
                    csipl_memory_hint hint);
```

Parameters

- N, integer scalar, input. Length of window.
- hint, enumerated type, input.

CSIPL_MEM_NONE	no hint
CSIPL_MEM_RDONLY	read-only
CSIPL_MEM_CONST	constant
CSIPL_MEM_SHARED	shared
CSIPL_MEM_SHARED_RDONLY	shared and read-only
CSIPL_MEM_SHARED_CONST	shared and constant

Return Value

- real vector.

Description

Creates a vector initialised with a Blackman window of length N.

$$X[k] := 0.42 - 0.5\cos\left(\frac{2\pi k}{N-1}\right) + 0.8\cos\left(\frac{4\pi k}{N-1}\right).$$

NULL is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. N > 1.
2. hint must be valid.

Notes

csipl_vcreate_cheby_f

Create a vector with Dolph-Chebyshev window weights.

Prototype

```
float  * csipl_vcreate_cheby_f(
                    unsigned int      N,
                    float             ripple,
                    csipl_memory_hint hint);
```

Parameters

- N, integer scalar, input. Length of window.
- ripple, real scalar, input. Side lobes are this number of decibels below the main lobe.
- hint, enumerated type, input.

CSIPL_MEM_NONE	no hint
CSIPL_MEM_RDONLY	read-only
CSIPL_MEM_CONST	constant
CSIPL_MEM_SHARED	shared
CSIPL_MEM_SHARED_RDONLY	shared and read-only
CSIPL_MEM_SHARED_CONST	shared and constant

Return Value

- real vector.

Description

Creates a vector initialised with a Dolph-Chebyshev window of length N.

$$\delta_p := 10^{-ripple/20}$$

$$\tau_p := \frac{1+\delta_p}{\delta_p}$$

$$\delta_f := \frac{1}{\pi}\ \cos^{-1}\left[\frac{1}{\cosh\left(\frac{\cosh^{-1}(\tau_p)}{N-1}\right)}\right]$$

$$x[0] := \frac{3-\cos(2\pi\delta_f)}{1+\cos(2\pi\delta_f)}$$

$$x[k] := \frac{x[0]+1}{2}\ \left(\cos\left(\frac{2\pi k}{N}\right)\right) + \frac{x[0]-1}{2}$$

N odd:

$$W[k] := \begin{cases} \delta_p \cosh\left(\frac{N-1}{2} \cosh^{-1}(x[k])\right) & : \quad |x[k]| > 1 \\ \delta_p \cos\left(\frac{N-1}{2} \cos^{-1}(x[k])\right) & : \quad |x[k]| \leqslant 1 \end{cases}$$

N even:

$$W[k] := \begin{cases} \delta_p \cosh\left(\frac{N-1}{2} \cosh^{-1}(x[k])\right) \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| > 1, \quad 0 \leqslant k \leqslant \lfloor N/2 \rfloor \\ -\delta_p \cosh\left(\frac{N-1}{2} \cosh^{-1}(x[k])\right) \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| > 1, \quad \lfloor N/2 \rfloor < k < N \\ \delta_p \cos\left(\frac{N-1}{2} \cos^{-1}(x[k])\right) \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| \leqslant 1, \quad 0 \leqslant k \leqslant \lfloor N/2 \rfloor \\ -\delta_p \cos\left(\frac{N-1}{2} \cos^{-1}(x[k])\right) \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| \leqslant 1, \quad \lfloor N/2 \rfloor < k < N. \end{cases}$$

FFT the W's:

$$w[k] := \sum_{j=0}^{N-1} W[k] \exp(2\pi \mathrm{i} jk/N).$$

Populate the window with the frequency swap of the w's:

$$X[k] := \begin{cases} \mathrm{real}\left(\frac{w[k+\lfloor N/2 \rfloor]}{w[0]}\right) & : \quad 0 \leqslant k < \lfloor N/2 \rfloor \\ \mathrm{real}\left(\frac{w[k-\lfloor N/2 \rfloor]}{w[0]}\right) & : \quad \lfloor N/2 \rfloor \leqslant k < N. \end{cases}$$

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `N` > 0.
2. `hint` must be a valid.

Notes

csipl_vcreate_hanning_f

Create a vector with Hanning window weights.

Prototype

```
float  * csipl_vcreate_hanning_f(
                    unsigned int      N,
                    csipl_memory_hint hint);
```

Parameters

- N, integer scalar, input. Length of window.
- hint, enumerated type, input.

CSIPL_MEM_NONE	no hint
CSIPL_MEM_RDONLY	read-only
CSIPL_MEM_CONST	constant
CSIPL_MEM_SHARED	shared
CSIPL_MEM_SHARED_RDONLY	shared and read-only
CSIPL_MEM_SHARED_CONST	shared and constant

Return Value

- real vector.

Description

Creates a vector initialised with a Hanning window of length N.

$$X[k] := 0.5\left(1 - \cos\left(\frac{2\pi(k+1)}{N+1}\right)\right).$$

NULL is returned if the create fails.

Restrictions

Restrictions

Errors

The arguments must conform to the following:

1. N > 1.
2. hint must be valid.

Notes

There are two different widely used definitions of the Hanning window. The other is

$$X[k] := 0.5\left(1 - \cos\left(\frac{2\pi k)}{N-1}\right)\right).$$

This form has a weight of zero for both end points of the window; we use the form that does not have zero end points.

If you want the window to be periodic of length N, you must generate a Hanning window of length $N-1$, copy it to a vector of length N, and set the last point to 0.0.

csipl_vcreate_kaiser_f

Create a vector with Kaiser window weights.

Prototype

```
float  * csipl_vcreate_kaiser_f(
                    unsigned int      N,
                    float             beta,
                    csipl_memory_hint hint);
```

Parameters

- N, integer scalar, input. Length of window.
- beta, real scalar, input. Transition width.
- hint, enumerated type, input.

CSIPL_MEM_NONE	no hint
CSIPL_MEM_RDONLY	read-only
CSIPL_MEM_CONST	constant
CSIPL_MEM_SHARED	shared
CSIPL_MEM_SHARED_RDONLY	shared and read-only
CSIPL_MEM_SHARED_CONST	shared and constant

Return Value

- real vector.

Description

Creates a vector initialised with a Kaiser window of length N.

$$X[k] := \frac{I_0\left[\beta\sqrt{1-\left(\frac{2k-N+1}{N-1}\right)^2}\right]}{I_0[\beta]} \text{ where } I_0[x] = \sum_{p=0}^{\infty}\left(\frac{x^p}{2^p\,p!}\right)^2.$$

Increasing β widens the main lobe (transition width) and reduces the side lobes.

NULL is returned if the create fails.

Restrictions

Errors

Notes

7.4 Filter Functions

- csipl_fir_create_f
- csipl_cfir_create_inter_f
- csipl_cfir_create_split_f
- csipl_Dfir_destroy_P
- csipl_firflt_f
- csipl_cfirflt_inter_f
- csipl_cfirflt_split_f
- csipl_Dfir_getattr_P
- csipl_Dfir_reset_P

csipl_fir_create_f

Create a decimated FIR filter object.

Prototype

```
csipl_fir_f * csipl_fir_create_f(
                    float            *kernel,
                    csipl_stride      stridekernel,
                    csipl_symmetry    symm,
                    unsigned int      N,
                    unsigned int      D,
                    csipl_obj_state   state,
                    unsigned int      ntimes,
                    csipl_alg_hint    hint,
                    csipl_length      n);
```

Parameters

- `kernel`, real vector, input. Vector of non-redundant filter coefficients. There are $M+1$ in the non-symmetric case and $\lceil (M+1)/2 \rceil$ for symmetric filters.
- `stridekernel`, integer scalar, input.
- `symm`, enumerated type, input.

`CSIPL_NONSYM`	non-symmetric
`CSIPL_SYM_EVEN_LEN_ODD`	(even) symmetric, odd length
`CSIPL_SYM_EVEN_LEN_EVEN`	(even) symmetric, even length

- `N`, integer scalar, input. Length of data vector.
- `D`, integer scalar, input. Decimation factor.
- `state`, enumerated type, input.

`CSIPL_STATE_NO_SAVE`	do not save state — single call filter
`CSIPL_STATE_SAVE`	save state for continuous filter

- `ntimes`, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

- `n`, integer scalar, input.

Return Value

- structure.

Description

Creates a decimated FIR filter object and returns a pointer to the object. The user specifies the kernel (filter coefficients and filter order), the integral output decimation factor, D, the length of the input segments (vectors) that will be filtered, and whether to save state information for continuous filtering.

If the create fails, `NULL` is returned.

If requested, the FIR filter object encapsulates the filter's state information. The state is initialised to zero. The filter state allows long data streams to be processed in segments by successive calls to `csipl_firflt_f`.

The FIR filter object is used to compute:

$$y[k] := \sum_{j=0}^{M} h[j]\,\widehat{x}[p+kD-j] \text{ where } \widehat{x}[j] = \begin{cases} s[j] & : \quad j < 0 \\ x[j] & : \quad j \geqslant 0 \end{cases}, \text{ and } 0 \leqslant k < \lceil (N-p)/D \rceil.$$

The vector s and integer p are private internal state information. When the FIR filter object is created they are initialised to zero, and they will remain so if the SAVE option is not specified. Otherwise
$s[j] := x[N+j]$ for $-M \leqslant j < 0$ and
$p := D - 1 - [(N-1-p) \bmod D]$.

Given a filter kernel of order M with coefficient vector h, segment length N, and decimation factor D, the decimated output y is of length $(N-p)/D$.

Restrictions

The decimation factor must be less than or equal to the filter length.

Errors

Notes

For non-lowpass filters, set $D = 1$.

It is important that the kernel vector be only as long as necessary (see above) — the symmetric values of the filter between the kernel's centre and its last value are not to be included in the kernel.

It is safe to destroy the kernel after creating the FIR filter object.

The filter will be evaluated directly unless `hint` is `CSIPL_ALG_TIME`, in which case convolution in the frequency domain (a method based on FFTs) will be used if it is quicker.

csipl_cfir_create_inter_f

Create a decimated FIR filter object.

Prototype

```
csipl_cfir_f * csipl_cfir_create_inter_f(
                    void              *kernel,
                    csipl_stride       stridekernel,
                    csipl_symmetry     symm,
                    unsigned int       N,
                    unsigned int       D,
                    csipl_obj_state    state,
                    unsigned int       ntimes,
                    csipl_alg_hint     hint,
                    csipl_length       n);
```

Parameters

- `kernel`, complex vector, input. Vector of non-redundant filter coefficients. There are $M + 1$ in the non-symmetric case and $\lceil (M + 1)/2 \rceil$ for symmetric filters.
- `stridekernel`, integer scalar, input.
- `symm`, enumerated type, input.

`CSIPL_NONSYM`	non-symmetric
`CSIPL_SYM_EVEN_LEN_ODD`	(even) symmetric, odd length
`CSIPL_SYM_EVEN_LEN_EVEN`	(even) symmetric, even length

- `N`, integer scalar, input. Length of data vector.
- `D`, integer scalar, input. Decimation factor.
- `state`, enumerated type, input.

`CSIPL_STATE_NO_SAVE`	do not save state — single call filter
`CSIPL_STATE_SAVE`	save state for continuous filter

- `ntimes`, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

- `n`, integer scalar, input.

Return Value

- structure.

Description

Creates a decimated FIR filter object and returns a pointer to the object. The user specifies the kernel (filter coefficients and filter order), the integral output decimation factor, D, the length of the input segments (vectors) that will be filtered, and whether to save state information for continuous filtering.

If the create fails, `NULL` is returned.

If requested, the FIR filter object encapsulates the filter's state information. The state is initialised to zero. The filter state allows long data streams to be processed in segments by successive calls to `csipl_cfirflt_f`.

The FIR filter object is used to compute:

$$y[k] := \sum_{j=0}^{M} h[j]\,\widehat{x}[p+kD-j] \text{ where } \widehat{x}[j] = \begin{cases} s[j] & : \quad j < 0 \\ x[j] & : \quad j \geqslant 0 \end{cases}, \text{ and } 0 \leqslant k < \lceil (N-p)/D \rceil.$$

The vector s and integer p are private internal state information. When the FIR filter object is created they are initialised to zero, and they will remain so if the SAVE option is not specified. Otherwise
$s[j] := x[N+j]$ for $-M \leqslant j < 0$ and
$p := D - 1 - [(N-1-p) \bmod D]$.

Given a filter kernel of order M with coefficient vector h, segment length N, and decimation factor D, the decimated output y is of length $(N-p)/D$.

Restrictions

The decimation factor must be less than or equal to the filter length.

Errors

Notes

For non-lowpass filters, set $D = 1$.

It is important that the kernel vector be only as long as necessary (see above) — the symmetric values of the filter between the kernel's centre and its last value are not to be included in the kernel.

It is safe to destroy the kernel after creating the FIR filter object.

The filter will be evaluated directly unless `hint` is `CSIPL_ALG_TIME`, in which case convolution in the frequency domain (a method based on FFTs) will be used if it is quicker.

csipl_cfir_create_split_f

Create a decimated FIR filter object.

Prototype

```
csipl_cfir_f * csipl_cfir_create_split_f(
                    float            *kernel_re,
                    float            *kernel_im,
                    csipl_stride      stridekernel,
                    csipl_symmetry    symm,
                    unsigned int      N,
                    unsigned int      D,
                    csipl_obj_state   state,
                    unsigned int      ntimes,
                    csipl_alg_hint    hint,
                    csipl_length      n);
```

Parameters

- `kernel_re`, real part of complex vector, length n, input.
- `kernel_im`, imaginary part of complex vector, length n, input. Vector of non-redundant filter coefficients. There are $M + 1$ in the non-symmetric case and $\lceil (M + 1)/2 \rceil$ for symmetric filters.
- `stridekernel`, integer scalar, input.
- `symm`, enumerated type, input.

`CSIPL_NONSYM`	non-symmetric
`CSIPL_SYM_EVEN_LEN_ODD`	(even) symmetric, odd length
`CSIPL_SYM_EVEN_LEN_EVEN`	(even) symmetric, even length

- `N`, integer scalar, input. Length of data vector.
- `D`, integer scalar, input. Decimation factor.
- `state`, enumerated type, input.

`CSIPL_STATE_NO_SAVE`	do not save state — single call filter
`CSIPL_STATE_SAVE`	save state for continuous filter

- `ntimes`, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.
- `hint`, enumerated type, input.

`CSIPL_ALG_SPACE`	minimise memory usage
`CSIPL_ALG_TIME`	minimise execution time
`CSIPL_ALG_NOISE`	maximise numerical accuracy

- `n`, integer scalar, input.

Return Value

- structure.

Description

Creates a decimated FIR filter object and returns a pointer to the object. The user specifies the kernel (filter coefficients and filter order), the integral output decimation factor, D, the length of the input segments (vectors) that will be filtered, and whether to save state information for continuous filtering.

If the create fails, `NULL` is returned.

If requested, the FIR filter object encapsulates the filter's state information. The state is initialised to zero. The filter state allows long data streams to be processed in segments by successive calls to `csipl_cfirflt_f`.

The FIR filter object is used to compute:

$$y[k] := \sum_{j=0}^{M} h[j]\,\widehat{x}[p+kD-j] \text{ where } \widehat{x}[j] = \begin{cases} s[j] & : \quad j < 0 \\ x[j] & : \quad j \geqslant 0 \end{cases}, \text{ and } 0 \leqslant k < \lceil (N-p)/D \rceil.$$

The vector s and integer p are private internal state information. When the FIR filter object is created they are initialised to zero, and they will remain so if the SAVE option is not specified. Otherwise
$s[j] := x[N+j]$ for $-M \leqslant j < 0$ and
$p := D - 1 - [(N-1-p) \bmod D]$.

Given a filter kernel of order M with coefficient vector h, segment length N, and decimation factor D, the decimated output y is of length $(N-p)/D$.

Restrictions

The decimation factor must be less than or equal to the filter length.

Errors

Notes

For non-lowpass filters, set $D = 1$.

It is important that the kernel vector be only as long as necessary (see above) — the symmetric values of the filter between the kernel's centre and its last value are not to be included in the kernel.

It is safe to destroy the kernel after creating the FIR filter object.

The filter will be evaluated directly unless `hint` is `CSIPL_ALG_TIME`, in which case convolution in the frequency domain (a method based on FFTs) will be used if it is quicker.

csipl_Dfir_destroy_P

Destroy a FIR filter object.

Prototype

```
int csipl_Dfir_destroy_P(
                    csipl_Dfir_P *plan);
```

The following instances are supported:

`csipl_fir_destroy_f`

`csipl_cfir_destroy_f`

Parameters

- `plan`, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) a FIR filter object. Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

An argument of `NULL` is not an error.

csipl_firflt_f

FIR filter an input sequence and decimate the output.

Prototype

```
int csipl_firflt_f(
                    csipl_fir_f   *plan,
                    float         *x,
                    csipl_stride   stridex,
                    float         *y,
                    csipl_stride   stridey,
                    csipl_length   n);
```

Parameters

- plan, structure, input.
- x, real vector, input.
- stridex, integer scalar, input.
- y, real vector, output.
- stridey, integer scalar, input.
- n, integer scalar, input.

Return Value

- number of values computed.

Description

Applies a FIR filter, specified by the FIR filter object, to an input segment x, and computes a decimated output segment y. Initial and final filter state is encapsulated in the FIR filter object. Long data streams can be processed in segments by successive calls to this function.

When N is a multiple of D, the length of y and the number of output samples is N/D.

When N is not a multiple of D, the length of y is $\lceil N/D \rceil$, although it may not be fully populated; there may be only $\lfloor N/D \rfloor$ values.

The return value is the number of output samples computed.

Restrictions

Filtering cannot be performed in place.

Errors

The arguments must conform to the following:

1. The FIR filter object must be valid.
2. The input vector `x` must be of length N (conformant with the FIR filter object).
3. The output vector `y` must be of length $\lceil N/D \rceil$ (conformant with the FIR filter object).
4. The input `x`, and the output `y`, must not overlap.

Notes

The filter object may be modified with the updated state.

csipl_cfirflt_inter_f

FIR filter an input sequence and decimate the output.

Prototype

```
int csipl_cfirflt_inter_f(
                    csipl_cfir_f  *plan,
                    void          *x,
                    csipl_stride   stridex,
                    void          *y,
                    csipl_stride   stridey,
                    csipl_length   n);
```

Parameters

- `plan`, structure, input.
- `x`, complex vector, input.
- `stridex`, integer scalar, input.
- `y`, complex vector, output.
- `stridey`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- number of values computed.

Description

Applies a FIR filter, specified by the FIR filter object, to an input segment x, and computes a decimated output segment y. Initial and final filter state is encapsulated in the FIR filter object. Long data streams can be processed in segments by successive calls to this function.

When N is a multiple of D, the length of y and the number of output samples is N/D.

When N is not a multiple of D, the length of y is $\lceil N/D \rceil$, although it may not be fully populated; there may be only $\lfloor N/D \rfloor$ values.

The return value is the number of output samples computed.

Restrictions

Filtering cannot be performed in place.

Errors

The arguments must conform to the following:

1. The FIR filter object must be valid.
2. The input vector `x` must be of length N (conformant with the FIR filter object).
3. The output vector `y` must be of length $\lceil N/D \rceil$ (conformant with the FIR filter object).
4. The input `x`, and the output `y`, must not overlap.

Notes

The filter object may be modified with the updated state.

csipl_cfirflt_split_f

FIR filter an input sequence and decimate the output.

Prototype

```
int csipl_cfirflt_split_f(
                    csipl_cfir_f  *plan,
                    float         *x_re,
                    float         *x_im,
                    csipl_stride   stridex,
                    float         *y_re,
                    float         *y_im,
                    csipl_stride   stridey,
                    csipl_length   n);
```

Parameters

- plan, structure, input.
- x_re, real part of complex vector, length n, input.
- x_im, imaginary part of complex vector, length n, input.
- stridex, integer scalar, input.
- y_re, real part of complex vector, length m, output.
- y_im, imaginary part of complex vector, length m, output.
- stridey, integer scalar, input.
- n, integer scalar, input.

Return Value

- number of values computed.

Description

Applies a FIR filter, specified by the FIR filter object, to an input segment x, and computes a decimated output segment y. Initial and final filter state is encapsulated in the FIR filter object. Long data streams can be processed in segments by successive calls to this function.

When N is a multiple of D, the length of y and the number of output samples is N/D.

When N is not a multiple of D, the length of y is $\lceil N/D \rceil$, although it may not be fully populated; there may be only $\lfloor N/D \rfloor$ values.

The return value is the number of output samples computed.

Restrictions

Filtering cannot be performed in place.

Errors

The arguments must conform to the following:

1. The FIR filter object must be valid.
2. The input vector `x` must be of length N (conformant with the FIR filter object).
3. The output vector `y` must be of length $\lceil N/D \rceil$ (conformant with the FIR filter object).
4. The input `x`, and the output `y`, must not overlap.

Notes

The filter object may be modified with the updated state.

csipl_Dfir_getattr_P

Return the attributes of a FIR filter object.

Prototype

```
void csipl_Dfir_getattr_P(
                    csipl_Dfir_P       *plan,
                    csipl_Dfir_attr_P  *attr);
```

The following instances are supported:

`csipl_fir_getattr_f`

`csipl_cfir_getattr_f`

Parameters

- `plan`, structure, input.
- `attr`, pointer to structure, output.
  The attribute structure contains the following information:

`unsigned int`	`kernel_len`	kernel length
`csipl_symmetry`	`symm`	kernel symmetry
`unsigned int`	`in_len`	filter input segment length
`unsigned int`	`out_len`	filter output segment length
`csipl_length`	`decimation`	decimation factor
`csipl_obj_state`	`state`	save state information

Return Value

- none.

Description

Returns the attributes of a FIR filter object.

Restrictions

Errors

The arguments must conform to the following:

1. The filter object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

The filter coefficient values are not accessible attributes.

csipl_Dfir_reset_P

Reset the state of a decimated FIR filter object.

Prototype

```
void csipl_Dfir_reset_P(
                    csipl_Dfir_P *fir);
```

The following instances are supported:

`csipl_fir_reset_f`

`csipl_cfir_reset_f`

Parameters

- `fir`, structure, input.

Return Value

- none.

Description

Resets the internal state of a previously created FIR filter object to the same state it had immediately after creation.

Restrictions

Errors

The arguments must conform to the following:

1. The filter object must be valid.

Notes

7.5 Miscellaneous Signal Processing Functions

- csipl_vhisto_f

csipl_vhisto_f

Compute the histogram of a vector.

Prototype

```
void csipl_vhisto_f(
                    float          *A,
                    csipl_stride    strideA,
                    float           min,
                    float           max,
                    csipl_hist_opt  opt,
                    float          *R,
                    csipl_stride    strideR,
                    csipl_length    n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- min, real scalar, input.
- max, real scalar, input.
- opt, enumerated type, input.

CSIPL_HIST_RESET	reset histogram each time
CSIPL_HIST_ACCUM	accumulate histogram

- R, real vector, output.
- strideR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Computes the histogram of a vector. Suppose the number of bins in the output vector is P. The first and last elements of the output vector are used to accumulate values outside the range of interest. The bin size is determined from the remaining $P-2$ bins and the boundary values.

The output vector is initialised to zero if the RESET option is used, otherwise the histogram is accumulated on top of the current data in the output vector.

The bin b is assigned as follows: if $\mathtt{A}[j*\mathtt{strideA}] < \mathtt{min}$ then $b := 0$ else if $\mathtt{A}[j*\mathtt{strideA}] \geqslant \mathtt{max}$ then $b := P - 1$ else $b := 1 + \lfloor (P-2)(\mathtt{A}[j*\mathtt{strideA}] - \mathtt{min})/(\mathtt{max} - \mathtt{min}) \rfloor$.

Restrictions

Errors

The arguments must conform to the following:

1. All the vector objects must be valid and of positive length.
2. $\mathtt{min} < \mathtt{max}$.

Notes

The first and last bins collect all the values less than `min`, and greater or equal to `max`, respectively.

Chapter 8. Linear Algebra

8.1 Matrix and Vector Operations

- csipl_cmherm_inter_f
- csipl_cmherm_split_f
- csipl_cvjdot_inter_f
- csipl_cvjdot_split_f
- csipl_gemp_f
- csipl_cgemp_inter_f
- csipl_cgemp_split_f
- csipl_gems_f
- csipl_cgems_inter_f
- csipl_cgems_split_f
- csipl_mprod_P
- csipl_cmprod_inter_P
- csipl_cmprod_split_P
- csipl_cmprodh_inter_P
- csipl_cmprodh_split_P
- csipl_cmprodj_inter_P
- csipl_cmprodj_split_P
- csipl_mprodt_P
- csipl_cmprodt_inter_P
- csipl_cmprodt_split_P
- csipl_mvprod_P
- csipl_cmvprod_inter_P
- csipl_cmvprod_split_P
- csipl_mtrans_P
- csipl_cmtrans_inter_P
- csipl_cmtrans_split_P
- csipl_vdot_f
- csipl_cvdot_inter_f
- csipl_cvdot_split_f
- csipl_vmprod_P

- csipl_cvmprod_inter_P
- csipl_cvmprod_split_P
- csipl_vouter_f
- csipl_cvouter_inter_f
- csipl_cvouter_split_f
- csipl_vcsummgval_inter_f
- csipl_vcsummgval_split_f
- csipl_minvlu_f
- csipl_cminvlu_inter_f
- csipl_cminvlu_split_f

csipl_cmherm_inter_f

Complex Hermitian (conjugate transpose) of a matrix.

Prototype

```
void csipl_cmherm_inter_f(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size n by m, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A}^{\mathrm{H}}$.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

csipl_cmherm_split_f

Complex Hermitian (conjugate transpose) of a matrix.

Prototype

```
void csipl_cmherm_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *R_re,
                    float        *R_im,
                    int           ldR,
                    csipl_length  m,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size n by m, output.
- R_im, imaginary part of complex matrix, size n by m, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A}^{\mathrm{H}}$.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

csipl_cvjdot_inter_f

Compute the conjugate inner (dot) product of two complex vectors.

Prototype

```
csipl_cscalar_f csipl_cvjdot_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value := $A^T \cdot B^*$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_cvjdot_split_f

Compute the conjugate inner (dot) product of two complex vectors.

Prototype

```
csipl_cscalar_f csipl_cvjdot_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value := $\mathtt{A}^{\mathrm{T}} \cdot \mathtt{B}^{*}$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_gemp_f

Calculate the general product of two matrices and accumulate.

Prototype

```
void csipl_gemp_f(
                    float          alpha,
                    float         *A,
                    int            ldA,
                    csipl_mat_op   Aop,
                    float         *B,
                    int            ldB,
                    csipl_mat_op   Bop,
                    float          beta,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

Parameters

- alpha, real scalar, input.
- A, real matrix, size m by p, input. The size shows the dimensions after Aop has been applied.
- ldA, integer scalar, input.
- Aop, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose

- B, real matrix, size p by n, input. The size shows the dimensions after Bop has been applied.
- ldB, integer scalar, input.
- Bop, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose

- beta, real scalar, input.
- R, real matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

- p, integer scalar, input.

Return Value

- none.

Description

R := alpha · Aop(A) · Bop(B) + beta · R
where Aop and Bop are matrix operations
(identity or transpose).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. Aop and Bop must be valid.

Notes

csipl_cgemp_inter_f

Calculate the general product of two matrices and accumulate.

Prototype

```
void csipl_cgemp_inter_f(
                    csipl_cscalar_f  alpha,
                    void            *A,
                    int              ldA,
                    csipl_mat_op     Aop,
                    void            *B,
                    int              ldB,
                    csipl_mat_op     Bop,
                    csipl_cscalar_f  beta,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n,
                    csipl_length     p);
```

Parameters

- `alpha`, complex scalar, input.
- `A`, complex matrix, size m by p, input. The size shows the dimensions after `Aop` has been applied.
- `ldA`, integer scalar, input.
- `Aop`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose
`CSIPL_MAT_HERM`	Hermitian (conjugate transpose)
`CSIPL_MAT_CONJ`	conjugate

- `B`, complex matrix, size p by n, input. The size shows the dimensions after `Bop` has been applied.
- `ldB`, integer scalar, input.
- `Bop`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose
`CSIPL_MAT_HERM`	Hermitian (conjugate transpose)
`CSIPL_MAT_CONJ`	conjugate

- `beta`, complex scalar, input.
- `R`, complex matrix, size m by n, output.

- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{alpha} \cdot \mathtt{Aop}(\mathtt{A}) \cdot \mathtt{Bop}(\mathtt{B}) + \mathtt{beta} \cdot \mathtt{R}$
where Aop and Bop are matrix operations
(identity, transpose, conjugate or Hermitian).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. Aop and Bop must be valid.

Notes

csipl_cgemp_split_f

Calculate the general product of two matrices and accumulate.

Prototype

```
void csipl_cgemp_split_f(
                    csipl_cscalar_f  alpha_re,
                    csipl_cscalar_f  alpha_im,
                    float           *A_re,
                    float           *A_im,
                    int              ldA,
                    csipl_mat_op     Aop,
                    float           *B_re,
                    float           *B_im,
                    int              ldB,
                    csipl_mat_op     Bop,
                    csipl_cscalar_f  beta_re,
                    csipl_cscalar_f  beta_im,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n,
                    csipl_length     p);
```

Parameters

- alpha_re, real part of complex scalar, input.
- alpha_im, imaginary part of complex scalar, input.
- A_re, real part of complex matrix, size m by p, input.
- A_im, imaginary part of complex matrix, size m by p, input. The size shows the dimensions after Aop has been applied.
- ldA, integer scalar, input.
- Aop, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose
CSIPL_MAT_HERM	Hermitian (conjugate transpose)
CSIPL_MAT_CONJ	conjugate

- B_re, real part of complex matrix, size p by n, input.
- B_im, imaginary part of complex matrix, size p by n, input. The size shows the dimensions after Bop has been applied.

- ldB, integer scalar, input.
- Bop, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose
CSIPL_MAT_HERM	Hermitian (conjugate transpose)
CSIPL_MAT_CONJ	conjugate

- beta_re, real part of complex scalar, input.
- beta_im, imaginary part of complex scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{alpha} \cdot \mathtt{Aop}(\mathtt{A}) \cdot \mathtt{Bop}(\mathtt{B}) + \mathtt{beta} \cdot \mathtt{R}$
where Aop and Bop are matrix operations

(identity, transpose, conjugate or Hermitian).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. Aop and Bop must be valid.

Notes

csipl_gems_f

Calculate a general matrix sum.

Prototype

```
void csipl_gems_f(
                    float          alpha,
                    float         *A,
                    int            ldA,
                    csipl_mat_op   Aop,
                    float          beta,
                    float         *C,
                    int            ldC,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- alpha, real scalar, input.
- A, real matrix, size m by n, input. The size shows the dimensions after Aop has been applied.
- ldA, integer scalar, input.
- Aop, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose

- beta, real scalar, input.
- C, real matrix, size m by n, output.
- ldC, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

C := alpha · Aop(A) + beta · C
where Aop is a matrix operation (identity or transpose).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` must be valid.

Notes

csipl_cgems_inter_f

Calculate a general matrix sum.

Prototype

```
void csipl_cgems_inter_f(
                    csipl_cscalar_f  alpha,
                    void            *A,
                    int              ldA,
                    csipl_mat_op     Aop,
                    csipl_cscalar_f  beta,
                    void            *C,
                    int              ldC,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- `alpha`, complex scalar, input.
- `A`, complex matrix, size m by n, input. The size shows the dimensions after `Aop` has been applied.
- `ldA`, integer scalar, input.
- `Aop`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose
`CSIPL_MAT_HERM`	Hermitian (conjugate transpose)
`CSIPL_MAT_CONJ`	conjugate

- `beta`, complex scalar, input.
- `C`, complex matrix, size m by n, output.
- `ldC`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\texttt{C} := \texttt{alpha} \cdot \texttt{Aop}(\texttt{A}) + \texttt{beta} \cdot \texttt{C}$
where `Aop` is a matrix operation (identity, transpose, conjugate or Hermitian).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` must be valid.

Notes

csipl_cgems_split_f

Calculate a general matrix sum.

Prototype

```
void csipl_cgems_split_f(
                    csipl_cscalar_f  alpha_re,
                    csipl_cscalar_f  alpha_im,
                    float           *A_re,
                    float           *A_im,
                    int              ldA,
                    csipl_mat_op     Aop,
                    csipl_cscalar_f  beta_re,
                    csipl_cscalar_f  beta_im,
                    float           *C_re,
                    float           *C_im,
                    int              ldC,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- alpha_re, real part of complex scalar, input.
- alpha_im, imaginary part of complex scalar, input.
- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input. The size shows the dimensions after Aop has been applied.
- ldA, integer scalar, input.
- Aop, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose
CSIPL_MAT_HERM	Hermitian (conjugate transpose)
CSIPL_MAT_CONJ	conjugate

- beta_re, real part of complex scalar, input.
- beta_im, imaginary part of complex scalar, input.
- C_re, real part of complex matrix, size m by n, output.
- C_im, imaginary part of complex matrix, size m by n, output.
- ldC, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\texttt{C} := \texttt{alpha} \cdot \texttt{Aop}(\texttt{A}) + \texttt{beta} \cdot \texttt{C}$
where `Aop` is a matrix operation (identity, transpose, conjugate or Hermitian).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` must be valid.

Notes

csipl_mprod_P

Calculate the product of two matrices.

Prototype

```
void csipl_mprod_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

- csipl_mprod_f
- csipl_mprod_i
- csipl_mprod_si

Parameters

- A, matrix, size m by p, input.
- ldA, integer scalar, input.
- B, matrix, size p by n, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

R := A · B.

Restrictions

Errors

Notes

csipl_cmprod_inter_P

Calculate the product of two matrices.

Prototype

```
void csipl_cmprod_inter_P(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

- csipl_cmprod_inter_f
- csipl_cmprod_inter_i
- csipl_cmprod_inter_si

Parameters

- A, complex matrix, size m by p, input.
- ldA, integer scalar, input.
- B, complex matrix, size p by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

R := A · B.

Restrictions

Errors

Notes

csipl_cmprod_split_P

Calculate the product of two matrices.

Prototype

```
void csipl_cmprod_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    int            ldB,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

`csipl_cmprod_split_f`

`csipl_cmprod_split_i`

`csipl_cmprod_split_si`

Parameters

- `A_re`, real part of complex matrix, size m by p, input.
- `A_im`, imaginary part of complex matrix, size m by p, input.
- `ldA`, integer scalar, input.
- `B_re`, real part of complex matrix, size p by n, input.
- `B_im`, imaginary part of complex matrix, size p by n, input.
- `ldB`, integer scalar, input.
- `R_re`, real part of complex matrix, size m by n, output.
- `R_im`, imaginary part of complex matrix, size m by n, output.
- `ldR`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}$.

Restrictions

Errors

Notes

csipl_cmprodh_inter_P

Calculate the product a complex matrix and the Hermitian of a complex matrix.

Prototype

```
void csipl_cmprodh_inter_P(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

```
csipl_cmprodh_inter_f
csipl_cmprodh_inter_i
csipl_cmprodh_inter_si
```

Parameters

- A, complex matrix, size m by p, input.
- ldA, integer scalar, input.
- B, complex matrix, size n by p, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}^{\mathrm{H}}$.

Restrictions

Errors

Notes

csipl_cmprodh_split_P

Calculate the product a complex matrix and the Hermitian of a complex matrix.

Prototype

```
void csipl_cmprodh_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    int            ldB,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

```
csipl_cmprodh_split_f
csipl_cmprodh_split_i
csipl_cmprodh_split_si
```

Parameters

- A_re, real part of complex matrix, size m by p, input.
- A_im, imaginary part of complex matrix, size m by p, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size n by p, input.
- B_im, imaginary part of complex matrix, size n by p, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}^{\mathrm{H}}$.

Restrictions

Errors

Notes

csipl_cmprodj_inter_P

Calculate the product a complex matrix and the conjugate of a complex matrix.

Prototype

```
void csipl_cmprodj_inter_P(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

- csipl_cmprodj_inter_f
- csipl_cmprodj_inter_i
- csipl_cmprodj_inter_si

Parameters

- A, complex matrix, size m by p, input.
- ldA, integer scalar, input.
- B, complex matrix, size p by n, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}^*$.

Restrictions

Errors

Notes

csipl_cmprodj_split_P

Calculate the product a complex matrix and the conjugate of a complex matrix.

Prototype

```
void csipl_cmprodj_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    int            ldB,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

```
csipl_cmprodj_split_f
csipl_cmprodj_split_i
csipl_cmprodj_split_si
```

Parameters

- A_re, real part of complex matrix, size m by p, input.
- A_im, imaginary part of complex matrix, size m by p, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size p by n, input.
- B_im, imaginary part of complex matrix, size p by n, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}^*$.

Restrictions

Errors

Notes

csipl_mprodt_P

Calculate the product of a matrix and the transpose of a matrix.

Prototype

```
void csipl_mprodt_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *B,
                    int            ldB,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

```
csipl_mprodt_f
csipl_mprodt_i
csipl_mprodt_si
```

Parameters

- A, matrix, size m by p, input.
- ldA, integer scalar, input.
- B, matrix, size n by p, input.
- ldB, integer scalar, input.
- R, matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}^{\mathrm{T}}$.

Restrictions

Errors

Notes

csipl_cmprodt_inter_P

Calculate the product of a matrix and the transpose of a matrix.

Prototype

```
void csipl_cmprodt_inter_P(
                    void          *A,
                    int            ldA,
                    void          *B,
                    int            ldB,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

```
csipl_cmprodt_inter_f
csipl_cmprodt_inter_i
csipl_cmprodt_inter_si
```

Parameters

- A, complex matrix, size m by p, input.
- ldA, integer scalar, input.
- B, complex matrix, size n by p, input.
- ldB, integer scalar, input.
- R, complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}^{\mathrm{T}}$.

Restrictions

Errors

Notes

csipl_cmprodt_split_P

Calculate the product of a matrix and the transpose of a matrix.

Prototype

```
void csipl_cmprodt_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *B_re,
                    scalar_P      *B_im,
                    int            ldB,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

The following instances are supported:

```
csipl_cmprodt_split_f
csipl_cmprodt_split_i
csipl_cmprodt_split_si
```

Parameters

- A_re, real part of complex matrix, size m by p, input.
- A_im, imaginary part of complex matrix, size m by p, input.
- ldA, integer scalar, input.
- B_re, real part of complex matrix, size n by p, input.
- B_im, imaginary part of complex matrix, size n by p, input.
- ldB, integer scalar, input.
- R_re, real part of complex matrix, size m by n, output.
- R_im, imaginary part of complex matrix, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A} \cdot \mathtt{B}^{\mathrm{T}}$.

Restrictions

Errors

Notes

csipl_mvprod_P

Calculate a matrix–vector product.

Prototype

```
void csipl_mvprod_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *X,
                    csipl_stride   strideX,
                    scalar_P      *Y,
                    csipl_stride   strideY,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mvprod_f
- csipl_mvprod_i
- csipl_mvprod_si

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- X, vector, length n, input.
- strideX, integer scalar, input.
- Y, vector, length m, output.
- strideY, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

Y := A · X.

Restrictions

Errors

Notes

csipl_cmvprod_inter_P

Calculate a matrix–vector product.

Prototype

```
void csipl_cmvprod_inter_P(
                    void          *A,
                    int            ldA,
                    void          *X,
                    csipl_stride   strideX,
                    void          *Y,
                    csipl_stride   strideY,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

`csipl_cmvprod_inter_f`

`csipl_cmvprod_inter_i`

`csipl_cmvprod_inter_si`

Parameters

- `A`, complex matrix, size m by n, input.
- `ldA`, integer scalar, input.
- `X`, complex vector, length n, input.
- `strideX`, integer scalar, input.
- `Y`, complex vector, length m, output.
- `strideY`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`Y` := `A` · `X`.

Restrictions

Errors

Notes

csipl_cmvprod_split_P

Calculate a matrix–vector product.

Prototype

```
void csipl_cmvprod_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *X_re,
                    scalar_P      *X_im,
                    csipl_stride   strideX,
                    scalar_P      *Y_re,
                    scalar_P      *Y_im,
                    csipl_stride   strideY,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cmvprod_split_f
csipl_cmvprod_split_i
csipl_cmvprod_split_si
```

Parameters

- `A_re`, real part of complex matrix, size m by n, input.
- `A_im`, imaginary part of complex matrix, size m by n, input.
- `ldA`, integer scalar, input.
- `X_re`, real part of complex vector, length n, input.
- `X_im`, imaginary part of complex vector, length n, input.
- `strideX`, integer scalar, input.
- `Y_re`, real part of complex vector, length m, output.
- `Y_im`, imaginary part of complex vector, length m, output.
- `strideY`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y} := \mathtt{A} \cdot \mathtt{X}$.

Restrictions

Errors

Notes

csipl_mtrans_P

Transpose a matrix.

Prototype

```
void csipl_mtrans_P(
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_mtrans_bl
- csipl_mtrans_f
- csipl_mtrans_i
- csipl_mtrans_si

Parameters

- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- R, matrix, size n by m, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{A}^{\mathrm{T}}$.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

csipl_cmtrans_inter_P

Transpose a matrix.

Prototype

```
void csipl_cmtrans_inter_P(
                    void          *A,
                    int            ldA,
                    void          *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_cmtrans_inter_f
- csipl_cmtrans_inter_i
- csipl_cmtrans_inter_si

Parameters

- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R, complex matrix, size n by m, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R := A^T.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

csipl_cmtrans_split_P

Transpose a matrix.

Prototype

```
void csipl_cmtrans_split_P(
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *R_re,
                    scalar_P      *R_im,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

csipl_cmtrans_split_f

csipl_cmtrans_split_i

csipl_cmtrans_split_si

Parameters

- A_re, real part of complex matrix, size m by n, input.
- A_im, imaginary part of complex matrix, size m by n, input.
- ldA, integer scalar, input.
- R_re, real part of complex matrix, size n by m, output.
- R_im, imaginary part of complex matrix, size n by m, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R $:=$ A^{T}.

Restrictions

If the matrix `A` is square, the transpose is in place if `A` and `R` resolve to the same object, otherwise `A` and `R` must be disjoint.

Errors

Notes

csipl_vdot_f

Compute the inner (dot) product of two vectors.

Prototype

```
float  csipl_vdot_f(
                    float         *A,
                    csipl_stride   strideA,
                    float         *B,
                    csipl_stride   strideB,
                    csipl_length   n);
```

Parameters

- A, real vector, length n, input.
- strideA, integer scalar, input.
- B, real vector, length n, input.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value := $\mathtt{A}^{\mathrm{T}} \cdot \mathtt{B}$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_cvdot_inter_f

Compute the inner (dot) product of two vectors.

Prototype

```
csipl_cscalar_f csipl_cvdot_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    void          *B,
                    csipl_stride   strideB,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- B, complex vector, length n, input.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value := $\mathtt{A}^{\mathrm{T}} \cdot \mathtt{B}$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_cvdot_split_f

Compute the inner (dot) product of two vectors.

Prototype

```
csipl_cscalar_f csipl_cvdot_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input.
- strideB, integer scalar, input.
- n, integer scalar, input.

Return Value

- complex scalar.

Description

return value := $\mathtt{A}^{\mathrm{T}} \cdot \mathtt{B}$.

Restrictions

Overflow may occur.

Errors

Notes

csipl_vmprod_P

Calculate a vector–matrix product.

Prototype

```
void csipl_vmprod_P(
                    scalar_P      *X,
                    csipl_stride   strideX,
                    scalar_P      *A,
                    int            ldA,
                    scalar_P      *Y,
                    csipl_stride   strideY,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_vmprod_f
- csipl_vmprod_i
- csipl_vmprod_si

Parameters

- X, vector, length m, input.
- strideX, integer scalar, input.
- A, matrix, size m by n, input.
- ldA, integer scalar, input.
- Y, vector, length n, output.
- strideY, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y} := \mathtt{X}^{\mathrm{T}} \cdot \mathtt{A}$.

Restrictions

Errors

Notes

csipl_cvmprod_inter_P

Calculate a vector–matrix product.

Prototype

```
void csipl_cvmprod_inter_P(
                    void          *X,
                    csipl_stride   strideX,
                    void          *A,
                    int            ldA,
                    void          *Y,
                    csipl_stride   strideY,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

- csipl_cvmprod_inter_f
- csipl_cvmprod_inter_i
- csipl_cvmprod_inter_si

Parameters

- X, complex vector, length m, input.
- strideX, integer scalar, input.
- A, complex matrix, size m by n, input.
- ldA, integer scalar, input.
- Y, complex vector, length n, output.
- strideY, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y} := \mathtt{X}^{\mathrm{T}} \cdot \mathtt{A}$.

Restrictions

Errors

Notes

csipl_cvmprod_split_P

Calculate a vector–matrix product.

Prototype

```
void csipl_cvmprod_split_P(
                    scalar_P      *X_re,
                    scalar_P      *X_im,
                    csipl_stride   strideX,
                    scalar_P      *A_re,
                    scalar_P      *A_im,
                    int            ldA,
                    scalar_P      *Y_re,
                    scalar_P      *Y_im,
                    csipl_stride   strideY,
                    csipl_length   m,
                    csipl_length   n);
```

The following instances are supported:

```
csipl_cvmprod_split_f
csipl_cvmprod_split_i
csipl_cvmprod_split_si
```

Parameters

- `X_re`, real part of complex vector, length m, input.
- `X_im`, imaginary part of complex vector, length m, input.
- `strideX`, integer scalar, input.
- `A_re`, real part of complex matrix, size m by n, input.
- `A_im`, imaginary part of complex matrix, size m by n, input.
- `ldA`, integer scalar, input.
- `Y_re`, real part of complex vector, length n, output.
- `Y_im`, imaginary part of complex vector, length n, output.
- `strideY`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

$\mathtt{Y} := \mathtt{X}^{\mathrm{T}} \cdot \mathtt{A}$.

Restrictions

Errors

Notes

csipl_vouter_f

Calculate the outer product of two vectors.

Prototype

```
void csipl_vouter_f(
                    float          alpha,
                    float         *X,
                    csipl_stride   strideX,
                    float         *Y,
                    csipl_stride   strideY,
                    float         *R,
                    int            ldR,
                    csipl_length   m,
                    csipl_length   n);
```

Parameters

- alpha, real scalar, input.
- X, real vector, length m, input.
- strideX, integer scalar, input.
- Y, real vector, length n, input.
- strideY, integer scalar, input.
- R, real vector, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{alpha} \cdot \mathtt{X} \cdot \mathtt{Y}^{\mathrm{T}}$.

Restrictions

Errors

Notes

csipl_cvouter_inter_f

Calculate the outer product of two vectors.

Prototype

```
void csipl_cvouter_inter_f(
                    csipl_cscalar_f  alpha,
                    void            *X,
                    csipl_stride     strideX,
                    void            *Y,
                    csipl_stride     strideY,
                    void            *R,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- alpha, complex scalar, input.
- X, complex vector, length m, input.
- strideX, integer scalar, input.
- Y, complex vector, length n, input.
- strideY, integer scalar, input.
- R, complex vector, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{alpha} \cdot \mathtt{X} \cdot \mathtt{Y}^{\mathrm{H}}$.

Restrictions

Errors

Notes

csipl_cvouter_split_f

Calculate the outer product of two vectors.

Prototype

```
void csipl_cvouter_split_f(
                    csipl_cscalar_f  alpha_re,
                    csipl_cscalar_f  alpha_im,
                    float           *X_re,
                    float           *X_im,
                    csipl_stride     strideX,
                    float           *Y_re,
                    float           *Y_im,
                    csipl_stride     strideY,
                    float           *R_re,
                    float           *R_im,
                    int              ldR,
                    csipl_length     m,
                    csipl_length     n);
```

Parameters

- alpha_re, real part of complex scalar, input.
- alpha_im, imaginary part of complex scalar, input.
- X_re, real part of complex vector, length m, input.
- X_im, imaginary part of complex vector, length m, input.
- strideX, integer scalar, input.
- Y_re, real part of complex vector, length n, input.
- Y_im, imaginary part of complex vector, length n, input.
- strideY, integer scalar, input.
- R_re, real part of complex vector, size m by n, output.
- R_im, imaginary part of complex vector, size m by n, output.
- ldR, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

$\mathtt{R} := \mathtt{alpha} \cdot \mathtt{X} \cdot \mathtt{Y}^{\mathrm{H}}$.

Restrictions

Errors

Notes

csipl_vcsummgval_inter_f

Returns the sum of the magnitudes of the elements of a complex vector.

Prototype

```
float  csipl_vcsummgval_inter_f(
                    void          *A,
                    csipl_stride   strideA,
                    csipl_length   n);
```

Parameters

- A, complex vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \sum |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_vcsummgval_split_f

Returns the sum of the magnitudes of the elements of a complex vector.

Prototype

```
float  csipl_vcsummgval_split_f(
                    float        *A_re,
                    float        *A_im,
                    csipl_stride  strideA,
                    csipl_length  n);
```

Parameters

- A_re, real part of complex vector, length n, input.
- A_im, imaginary part of complex vector, length n, input.
- strideA, integer scalar, input.
- n, integer scalar, input.

Return Value

- real scalar.

Description

return value $:= \sum |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

csipl_minvlu_f

Invert a square matrix using LU decomposition.

Prototype

```
void csipl_minvlu_f(
                    float         *A,
                    int            ldA,
                    signed int    *V,
                    csipl_stride   strideV,
                    float         *R,
                    int            ldR,
                    csipl_length   n);
```

Parameters

- A, real matrix, size n by n, input.
- ldA, integer scalar, input.
- V, integer vector, length n, input.
- strideV, integer scalar, input.
- R, real matrix, size n by n, output.
- ldR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R $:=$ A^{-1} where $0 \leqslant j < n$.

V is a list of pivots. It must not be null.

On return, all values will be positive if the inversion is successful; the first element will be -1 if the input matrix is singular or -2 if a memory allocation failed.

Restrictions

Errors

Notes

csipl_cminvlu_inter_f

Invert a square matrix using LU decomposition.

Prototype

```
void csipl_cminvlu_inter_f(
                    void          *A,
                    int            ldA,
                    signed int    *V,
                    csipl_stride   strideV,
                    void          *R,
                    int            ldR,
                    csipl_length   n);
```

Parameters

- A, complex matrix, size n by n, input.
- ldA, integer scalar, input.
- V, integer vector, length n, input.
- strideV, integer scalar, input.
- R, complex matrix, size n by n, output.
- ldR, integer scalar, input.
- n, integer scalar, input.

Return Value

- none.

Description

R $:=$ A^{-1} where $0 \leqslant j < n$.

V is a list of pivots. It must not be null.

On return, all values will be positive if the inversion is successful; the first element will be -1 if the input matrix is singular or -2 if a memory allocation failed.

Restrictions

Errors

Notes

csipl_cminvlu_split_f

Invert a square matrix using LU decomposition.

Prototype

```
void csipl_cminvlu_split_f(
                    float         *A_re,
                    float         *A_im,
                    int            ldA,
                    signed int    *V,
                    csipl_stride   strideV,
                    float         *R_re,
                    float         *R_im,
                    int            ldR,
                    csipl_length   n);
```

Parameters

- `A_re`, real part of complex matrix, size n by n, input.
- `A_im`, imaginary part of complex matrix, size n by n, input.
- `ldA`, integer scalar, input.
- `V`, integer vector, length n, input.
- `strideV`, integer scalar, input.
- `R_re`, real part of complex matrix, size n by n, output.
- `R_im`, imaginary part of complex matrix, size n by n, output.
- `ldR`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- none.

Description

`R` $:=$ `A`$^{-1}$ where $0 \leqslant j < n$.

`V` is a list of pivots. It must not be null.

On return, all values will be positive if the inversion is successful; the first element will be -1 if the input matrix is singular or -2 if a memory allocation failed.

Restrictions

Errors

Notes

8.2 Special Linear System Solvers

- csipl_covsol_f
- csipl_ccovsol_inter_f
- csipl_ccovsol_split_f
- csipl_llsqsol_f
- csipl_cllsqsol_inter_f
- csipl_cllsqsol_split_f
- csipl_toepsol_f
- csipl_ctoepsol_inter_f
- csipl_ctoepsol_split_f

csipl_covsol_f

Solve a covariance linear system problem.

Prototype

```
int csipl_covsol_f(
                    float        *A,
                    int           ldA,
                    float        *XB,
                    int           ldXB,
                    csipl_length  m,
                    csipl_length  n,
                    csipl_length  p);
```

Parameters

- A, real matrix, size m by n, input. The matrix A.
- ldA, integer scalar, input.
- XB, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: A is not of full rank.

Description

Solves the covariance linear system problem $A^{\mathrm{T}}AX = B$ where A is an m by n matrix of rank n and B is an n by p matrix, and $n \leqslant m$.

Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A may be overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.

The matrix A is assumed to be of full rank. This property is checked and a positive return value indicates that an error occurred.

csipl_ccovsol_inter_f

Solve a covariance linear system problem.

Prototype

```
int csipl_ccovsol_inter_f(
                    void          *A,
                    int            ldA,
                    void          *XB,
                    int            ldXB,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

Parameters

- A, complex matrix, size m by n, input. The matrix A.
- ldA, integer scalar, input.
- XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: A is not of full rank.

Description

Solves the covariance linear system problem $A^{\mathrm{H}}AX = B$ where A is an m by n matrix of rank n and B is an n by p matrix, and $n \leqslant m$.

Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A may be overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.

The matrix A is assumed to be of full rank. This property is checked and a positive return value indicates that an error occurred.

csipl_ccovsol_split_f

Solve a covariance linear system problem.

Prototype

```
int csipl_ccovsol_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *XB_re,
                    float        *XB_im,
                    int           ldXB,
                    csipl_length  m,
                    csipl_length  n,
                    csipl_length  p);
```

Parameters

- `A_re`, real part of complex matrix, size m by n, input.
- `A_im`, imaginary part of complex matrix, size m by n, input. The matrix A.
- `ldA`, integer scalar, input.
- `XB_re`, real part of complex matrix, size n by p, modified in place.
- `XB_im`, imaginary part of complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- `ldXB`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: `A` is not of full rank.

Description

Solves the covariance linear system problem $A^{\mathrm{H}}AX = B$ where A is an m by n matrix of rank n and B is an n by p matrix, and $n \leqslant m$.

Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A may be overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.

The matrix A is assumed to be of full rank. This property is checked and a positive return value indicates that an error occurred.

csipl_llsqsol_f

Solve a linear least squares problem.

Prototype

```
int csipl_llsqsol_f(
                    float        *A,
                    int           ldA,
                    float        *XB,
                    int           ldXB,
                    csipl_length  m,
                    csipl_length  n,
                    csipl_length  p);
```

Parameters

- A, real matrix, size m by n, input. The matrix A.
- ldA, integer scalar, input.
- XB, real matrix, size m by p, modified in place. On input, the matrix B. On output, the first n rows contain the matrix X.
- ldXB, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: A is not of full rank.

Description

Solves the linear least squares problem $\min||AX - B||_2$ where A is an m by n matrix of rank n and B is an m by p matrix, and $n \leqslant m$.

Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A is overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.

The matrix A is assumed to be of full rank. This property is checked. A positive return value indicates that the matrix did not have full column rank and the algorithm failed to be completed.

csipl_cllsqsol_inter_f

Solve a linear least squares problem.

Prototype

```
int csipl_cllsqsol_inter_f(
                    void          *A,
                    int            ldA,
                    void          *XB,
                    int            ldXB,
                    csipl_length   m,
                    csipl_length   n,
                    csipl_length   p);
```

Parameters

- A, complex matrix, size m by n, input. The matrix A.
- ldA, integer scalar, input.
- XB, complex matrix, size m by p, modified in place. On input, the matrix B. On output, the first n rows contain the matrix X.
- ldXB, integer scalar, input.
- m, integer scalar, input.
- n, integer scalar, input.
- p, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: A is not of full rank.

Description

Solves the linear least squares problem $\min||AX - B||_2$ where A is an m by n matrix of rank n and B is an m by p matrix, and $n \leqslant m$.

Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A is overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.

The matrix A is assumed to be of full rank. This property is checked. A positive return value indicates that the matrix did not have full column rank and the algorithm failed to be completed.

csipl_cllsqsol_split_f

Solve a linear least squares problem.

Prototype

```
int csipl_cllsqsol_split_f(
                    float        *A_re,
                    float        *A_im,
                    int           ldA,
                    float        *XB_re,
                    float        *XB_im,
                    int           ldXB,
                    csipl_length  m,
                    csipl_length  n,
                    csipl_length  p);
```

Parameters

- `A_re`, real part of complex matrix, size m by n, input.
- `A_im`, imaginary part of complex matrix, size m by n, input. The matrix A.
- `ldA`, integer scalar, input.
- `XB_re`, real part of complex matrix, size m by p, modified in place.
- `XB_im`, imaginary part of complex matrix, size m by p, modified in place. On input, the matrix B. On output, the first n rows contain the matrix X.
- `ldXB`, integer scalar, input.
- `m`, integer scalar, input.
- `n`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: `A` is not of full rank.

Description

Solves the linear least squares problem $\min||AX - B||_2$ where A is an m by n matrix of rank n and B is an m by p matrix, and $n \leqslant m$.

Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A is overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.

The matrix A is assumed to be of full rank. This property is checked. A positive return value indicates that the matrix did not have full column rank and the algorithm failed to be completed.

csipl_toepsol_f

Solve a real symmetric positive definite Toeplitz linear system.

Prototype

```
int csipl_toepsol_f(
                    float         *T,
                    csipl_stride   strideT,
                    float         *B,
                    csipl_stride   strideB,
                    float         *W,
                    csipl_stride   strideW,
                    float         *X,
                    csipl_stride   strideX,
                    csipl_length   n);
```

Parameters

- T, real vector, length n, input. First row of the Toeplitz matrix T.
- strideT, integer scalar, input.
- B, real vector, length n, input. The vector B.
- strideB, integer scalar, input.
- W, real vector, length n, modified in place. Workspace.
- strideW, integer scalar, input.
- X, real vector, length n, output. The vector x.
- strideX, integer scalar, input.
- n, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: T is not positive definite.

Description

Solves the real symmetric positive definite Toeplitz linear system $Tx = B$ where

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & t_{n-2} & t_{n-1} \\ t_1 & t_0 & t_1 & & t_{n-2} \\ \vdots & t_1 & \ddots & \ddots & \vdots \\ t_{n-2} & & \ddots & \ddots & t_1 \\ t_{n-1} & t_{n-2} & \cdots & t_1 & t_0 \end{bmatrix}.$$

We only need a vector containing the first row of T to specify the system.

Returns zero on success, -1 on memory allocation failure, and a positive value if T is not positive definite.

Restrictions

Errors

Notes

The matrix T is assumed to be of full rank and positive definite. This property is not checked. A positive return value indicates that an error occurred and the algorithm failed to be completed.

csipl_ctoepsol_inter_f

Solve a complex Hermitian positive definite Toeplitz linear system.

Prototype

```
int csipl_ctoepsol_inter_f(
                    void          *T,
                    csipl_stride   strideT,
                    void          *B,
                    csipl_stride   strideB,
                    void          *W,
                    csipl_stride   strideW,
                    void          *X,
                    csipl_stride   strideX,
                    csipl_length   n);
```

Parameters

- `T`, complex vector, length n, input. First row of the Toeplitz matrix T.
- `strideT`, integer scalar, input.
- `B`, complex vector, length n, input. The vector B.
- `strideB`, integer scalar, input.
- `W`, complex vector, length n, modified in place. Workspace.
- `strideW`, integer scalar, input.
- `X`, complex vector, length n, output. The vector x.
- `strideX`, integer scalar, input.
- `n`, integer scalar, input.

Return Value

- 0 : success.
- -1 : out of memory.
- $\geqslant 1$: `T` is not positive definite.

Description

Solves the complex Hermitian positive definite Toeplitz linear system $Tx = B$ where

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & t_{n-2} & t_{n-1} \\ t_1^* & t_0 & t_1 & & t_{n-2} \\ \vdots & t_1^* & \ddots & \ddots & \vdots \\ t_{n-2}^* & & \ddots & \ddots & t_1 \\ t_{n-1}^* & t_{n-2}^* & \cdots & t_1^* & t_0 \end{bmatrix}.$$

We only need a vector containing the first row of T to specify the system.

Returns zero on success, -1 on memory allocation failure, and a positive value if T is not positive definite.

Restrictions

Errors

Notes

The matrix T is assumed to be of full rank and positive definite. This property is not checked. A positive return value indicates that an error occurred and the algorithm failed to be completed.

csipl_ctoepsol_split_f

Solve a complex Hermitian positive definite Toeplitz linear system.

Prototype

```
int csipl_ctoepsol_split_f(
                    float        *T_re,
                    float        *T_im,
                    csipl_stride  strideT,
                    float        *B_re,
                    float        *B_im,
                    csipl_stride  strideB,
                    float        *W_re,
                    float        *W_im,
                    csipl_stride  strideW,
                    float        *X_re,
                    float        *X_im,
                    csipl_stride  strideX,
                    csipl_length  n);
```

Parameters

- T_re, real part of complex vector, length n, input.
- T_im, imaginary part of complex vector, length n, input. First row of the Toeplitz matrix T.
- strideT, integer scalar, input.
- B_re, real part of complex vector, length n, input.
- B_im, imaginary part of complex vector, length n, input. The vector B.
- strideB, integer scalar, input.
- W_re, real part of complex vector, length n, modified in place.
- W_im, imaginary part of complex vector, length n, modified in place. Workspace.
- strideW, integer scalar, input.
- X_re, real part of complex vector, length n, output.
- X_im, imaginary part of complex vector, length n, output. The vector x.
- strideX, integer scalar, input.
- n, integer scalar, input.

Return Value

- 0 : success.

- -1 : out of memory.
- $\geqslant 1$: T is not positive definite.

Description

Solves the complex Hermitian positive definite Toeplitz linear system $Tx = B$ where

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & t_{n-2} & t_{n-1} \\ t_1^* & t_0 & t_1 & & t_{n-2} \\ \vdots & t_1^* & \ddots & \ddots & \vdots \\ t_{n-2}^* & & \ddots & \ddots & t_1 \\ t_{n-1}^* & t_{n-2}^* & \cdots & t_1^* & t_0 \end{bmatrix}.$$

We only need a vector containing the first row of T to specify the system.

Returns zero on success, -1 on memory allocation failure, and a positive value if T is not positive definite.

Restrictions

Errors

Notes

The matrix T is assumed to be of full rank and positive definite. This property is not checked. A positive return value indicates that an error occurred and the algorithm failed to be completed.

8.3 General Square Linear System Solver

- csipl_lud_f
- csipl_clud_inter_f
- csipl_clud_split_f
- csipl_Dlud_create_P
- csipl_Dlud_destroy_P
- csipl_Dlud_getattr_P
- csipl_lusol_f
- csipl_clusol_inter_f
- csipl_clusol_split_f

csipl_lud_f

Compute an LU decomposition of a square matrix using partial pivoting.

Prototype

```
int csipl_lud_f(
                    csipl_clu_f *lud,
                    float       *A,
                    int          ldA);
```

Parameters

- `lud`, structure, input.
- `A`, real matrix, size n by n, modified in place.
- `ldA`, integer scalar, input.

Return Value

- Error code.

Description

Computes the LU decomposition of a general square matrix A using partial pivoting with row interchanges.

An LU decomposition is a factorisation of the form $A = PLU$ where P is a permutation matrix, L is lower triangular, and U is upper triangular.

Returns zero on success, and non-zero if A does not have full rank.

Restrictions

The matrix A is overwritten by the decomposition, and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero pivot element was encountered.

csipl_clud_inter_f

Compute an LU decomposition of a square matrix using partial pivoting.

Prototype

```
int csipl_clud_inter_f(
                    csipl_clu_f *lud,
                    void        *A,
                    int          ldA);
```

Parameters

- `lud`, structure, input.
- `A`, complex matrix, size n by n, modified in place.
- `ldA`, integer scalar, input.

Return Value

- Error code.

Description

Computes the LU decomposition of a general square matrix A using partial pivoting with row interchanges.

An LU decomposition is a factorisation of the form $A = PLU$ where P is a permutation matrix, L is lower triangular, and U is upper triangular.

Returns zero on success, and non-zero if A does not have full rank.

Restrictions

The matrix A is overwritten by the decomposition, and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero pivot element was encountered.

csipl_clud_split_f

Compute an LU decomposition of a square matrix using partial pivoting.

Prototype

```
int csipl_clud_split_f(
                    csipl_clu_f *lud,
                    float       *A_re,
                    float       *A_im,
                    int          ldA);
```

Parameters

- lud, structure, input.
- A_re, real part of complex matrix, size n by n, modified in place.
- A_im, imaginary part of complex matrix, size n by n, modified in place.
- ldA, integer scalar, input.

Return Value

- Error code.

Description

Computes the LU decomposition of a general square matrix A using partial pivoting with row interchanges.

An LU decomposition is a factorisation of the form $A = PLU$ where P is a permutation matrix, L is lower triangular, and U is upper triangular.

Returns zero on success, and non-zero if A does not have full rank.

Restrictions

The matrix A is overwritten by the decomposition, and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero pivot element was encountered.

csipl_Dlud_create_P

Create an LU decomposition object.

Prototype

```
csipl_Dlu_P * csipl_Dlud_create_P(
                    unsigned int  N);
```

The following instances are supported:

csipl_lud_create_f

csipl_clud_create_f

Parameters

- N, integer scalar, input.

Return Value

- structure.

Description

Creates an LU decomposition object. The LU decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.

The LU decomposition object is used to compute the LU decomposition of a general square matrix A using partial pivoting with row interchanges.

An LU decomposition is a factorisation of the form $A = PLU$ where P is a permutation matrix, L is lower triangular, and U is upper triangular.

NULL is returned if the create fails.

Restrictions

Errors

The input parameter must conform to the following:

1. N is positive.

Notes

csipl_Dlud_destroy_P

Destroy an LU decomposition object.

Prototype

```
int csipl_Dlud_destroy_P(
                    csipl_Dlu_P *lud);
```

The following instances are supported:

```
csipl_lud_destroy_f
csipl_clud_destroy_f
```

Parameters

- `lud`, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) an LU decomposition object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The input argument must conform to the following:

1. The LU decomposition object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

csipl_Dlud_getattr_P

Returns the attributes of an LU decomposition object.

Prototype

```
void csipl_Dlud_getattr_P(
                    csipl_Dlu_P       *lud,
                    csipl_Dlu_attr_P  *attr);
```

The following instances are supported:

`csipl_lud_getattr_f`

`csipl_clud_getattr_f`

Parameters

- `lud`, structure, input.
- `attr`, pointer to structure, output.
  The attribute structure contains the following information:

`csipl_length`	`n`	number of rows and columns in matrix

Return Value

- none.

Description

Returns the attributes of an LU decomposition object.

Restrictions

Errors

The arguments must conform to the following:

1. The LU decomposition object `lud` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

csipl_lusol_f

Solve a square linear system.

Prototype

```
int csipl_lusol_f(
                    csipl_clu_f   *clud,
                    csipl_mat_op   opA,
                    float         *XB,
                    int            ldXB,
                    csipl_length   p);
```

Parameters

- `clud`, structure, input. An LU decomposition object for the matrix A.
- `opA`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose

- `XB`, real matrix, size n by p, modified in place. (n is implicit, provided by `clud`.) On input, the matrix B. On output, the matrix X.
- `ldXB`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- Error code.

Description

Solve the linear system $\mathrm{op}(A)X = B$ where op() is either the identity or transpose operation, for a general matrix A using the decomposition computed by `csipl_lud_f`

A is an n by n matrix of rank n and B is an n by p matrix.

Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after `csipl_lud_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_clusol_inter_f

Solve a square linear system.

Prototype

```
int csipl_clusol_inter_f(
                    csipl_clu_f   *clud,
                    csipl_mat_op   opA,
                    void          *XB,
                    int            ldXB,
                    csipl_length   p);
```

Parameters

- `clud`, structure, input. An LU decomposition object for the matrix A.
- `opA`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_HERM`	Hermitian (conjugate transpose)

- `XB`, complex matrix, size n by p, modified in place. (n is implicit, provided by `clud`.) On input, the matrix B. On output, the matrix X.
- `ldXB`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- Error code.

Description

Solve the linear system $\mathrm{op}(A)X = B$ where op() is either the identity or Hermitian operation, for a general matrix A using the decomposition computed by `csipl_clud_f`

A is an n by n matrix of rank n and B is an n by p matrix.

Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after `csipl_clud_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_clusol_split_f

Solve a square linear system.

Prototype

```
int csipl_clusol_split_f(
                    csipl_clu_f   *clud_re,
                    csipl_clu_f   *clud_im,
                    csipl_mat_op   opA,
                    float         *XB_re,
                    float         *XB_im,
                    int            ldXB,
                    csipl_length   p);
```

Parameters

- `clud_re`, real part of structure, input.
- `clud_im`, imaginary part of structure, input. An LU decomposition object for the matrix A.
- `opA`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_HERM`	Hermitian (conjugate transpose)

- `XB_re`, real part of complex matrix, size n by p, modified in place.
- `XB_im`, imaginary part of complex matrix, size n by p, modified in place. (n is implicit, provided by `clud`.) On input, the matrix B. On output, the matrix X.
- `ldXB`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- Error code.

Description

Solve the linear system $\mathrm{op}(A)X = B$ where op() is either the identity or Hermitian operation, for a general matrix A using the decomposition computed by `csipl_clud_f`

A is an n by n matrix of rank n and B is an n by p matrix.

Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after `csipl_clud_f`. fails. This will result in a non-zero unsuccessful return value.

8.4 Symmetric Positive Definite Linear System Solver

- csipl_chold_f
- csipl_cchold_inter_f
- csipl_cchold_split_f
- csipl_chold_create_f
- csipl_cchold_create_f
- csipl_Dchold_destroy_P
- csipl_Dchold_getattr_P
- csipl_cholsol_f
- csipl_ccholsol_inter_f
- csipl_ccholsol_split_f

csipl_chold_f

Compute a Cholesky decomposition of a symmetric positive definite matrix.

Prototype

```
int csipl_chold_f(
                    csipl_cchol_f *chold,
                    float         *A,
                    int            ldA);
```

Parameters

- `chold`, structure, input.
- `A`, real matrix, size n by n, modified in place.
- `ldA`, integer scalar, input.

Return Value

- Error code.

Description

The Cholesky decomposition of a symmetric positive definite n by n matrix A is given by $A = LL^{\mathrm{T}}$ where L is a lower triangular matrix.

There is not a utility function for accessing the factors.

Returns zero on success. The routine will fail if a leading minor is not positive definite.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the decomposition is required.

Errors

The input and input/output objects must conform to the following:

1. All objects must be valid.
2. The matrix `A` and the Cholesky decomposition object `chold` must be conformant.

Notes

The matrix A is assumed to be symmetric. This property is not checked. There is no reduced storage format for symmetric matrices so the full matrix must be specified. However, only half the matrix is referenced and modified.

csipl_cchold_inter_f

Compute a Cholesky decomposition of a Hermitian positive definite matrix.

Prototype

```
int csipl_cchold_inter_f(
                    csipl_cchol_f  *chold,
                    void           *A,
                    int             ldA);
```

Parameters

- `chold`, structure, input.
- `A`, complex matrix, size n by n, modified in place.
- `ldA`, integer scalar, input.

Return Value

- Error code.

Description

The Cholesky decomposition of a Hermitian positive definite n by n matrix A is given by $A = LL^{\mathrm{H}}$ where L is a lower triangular matrix.

There is not a utility function for accessing the factors.

Returns zero on success. The routine will fail if a leading minor is not positive definite.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the decomposition is required.

Errors

The input and input/output objects must conform to the following:

1. All objects must be valid.
2. The matrix `A` and the Cholesky decomposition object `chold` must be conformant.

Notes

The matrix A is assumed to be Hermitian. This property is not checked. There is no reduced storage format for Hermitian matrices so the full matrix must be specified. However, only half the matrix is referenced and modified.

csipl_cchold_split_f

Compute a Cholesky decomposition of a Hermitian positive definite matrix.

Prototype

```
int csipl_cchold_split_f(
                    csipl_cchol_f *chold,
                    float         *A_re,
                    float         *A_im,
                    int            ldA);
```

Parameters

- `chold`, structure, input.
- `A_re`, real part of complex matrix, size n by n, modified in place.
- `A_im`, imaginary part of complex matrix, size n by n, modified in place.
- `ldA`, integer scalar, input.

Return Value

- Error code.

Description

The Cholesky decomposition of a Hermitian positive definite n by n matrix A is given by $A = LL^{\mathrm{H}}$ where L is a lower triangular matrix.

There is not a utility function for accessing the factors.

Returns zero on success. The routine will fail if a leading minor is not positive definite.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the decomposition is required.

Errors

The input and input/output objects must conform to the following:

1. All objects must be valid.
2. The matrix `A` and the Cholesky decomposition object `chold` must be conformant.

Notes

The matrix A is assumed to be Hermitian. This property is not checked. There is no reduced storage format for Hermitian matrices so the full matrix must be specified. However, only half the matrix is referenced and modified.

csipl_chold_create_f

Creates a Cholesky decomposition object.

Prototype

```
csipl_chol_f * csipl_chold_create_f(
                    csipl_mat_uplo uplo,
                    unsigned int   n);
```

Parameters

- uplo, enumerated type, input.

CSIPL_TR_LOW	lower triangle
CSIPL_TR_UPP	upper triangle

- n, integer scalar, input.

Return Value

- structure.

Description

Creates a Cholesky decomposition object. The Cholesky decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.

The Cholesky decomposition object is used to compute the Cholesky decomposition of a symmetric positive definite n by n matrix A.

The Cholesky decomposition of a symmetric positive definite n by n matrix A is given by $A = LL^{\mathrm{T}}$ where L is a lower triangular matrix.

There is not a utility function for accessing the factors.

NULL is returned if the create fails.

Restrictions

Errors

The input parameters must conform to the following:

1. n is positive.
2. uplo is valid.

Notes

csipl_cchold_create_f

Creates a Cholesky decomposition object.

Prototype

```
csipl_cchol_f * csipl_cchold_create_f(
                    csipl_mat_uplo uplo,
                    unsigned int   n);
```

Parameters

- uplo, enumerated type, input.

CSIPL_TR_LOW	lower triangle
CSIPL_TR_UPP	upper triangle

- n, integer scalar, input.

Return Value

- structure.

Description

Creates a Cholesky decomposition object. The Cholesky decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.

The Cholesky decomposition object is used to compute the Cholesky decomposition of a Hermitian positive definite n by n matrix A.

The Cholesky decomposition of a Hermitian positive definite n by n matrix A is given by $A = LL^{\mathrm{H}}$ where L is a lower triangular matrix.

There is not a utility function for accessing the factors.

NULL is returned if the create fails.

Restrictions

Errors

The input parameters must conform to the following:

1. n is positive.
2. uplo is valid.

Notes

csipl_Dchold_destroy_P

Destroy a Cholesky decomposition object.

Prototype

```
int csipl_Dchold_destroy_P(
                    csipl_Dchol_P *chold);
```

The following instances are supported:

csipl_chold_destroy_f

csipl_cchold_destroy_f

Parameters

- chold, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) a Cholesky decomposition object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The input object must conform to the following:

1. The Cholesky decomposition object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

csipl_Dchold_getattr_P

Returns the attributes of a Cholesky decomposition object.

Prototype

```
void csipl_Dchold_getattr_P(
                    csipl_Dchol_P       *chold,
                    csipl_Dchol_attr_P  *attr);
```

The following instances are supported:

```
csipl_chold_getattr_f
csipl_cchold_getattr_f
```

Parameters

- `chold`, structure, input.
- `attr`, pointer to structure, output.
  The attribute structure contains the following information:

`csipl_mat_uplo`	`uplo`	upper or lower triangular matrix
`csipl_length`	`n`	number of rows and columns in matrix

Return Value

- none.

Description

Returns the attributes of a Cholesky decomposition object.

Restrictions

Errors

The input and output arguments must conform to the following:

1. The Cholesky decomposition object `chold` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

csipl_cholsol_f

Solve a symmetric positive definite linear system.

Prototype

```
int csipl_cholsol_f(
                    csipl_cchol_f *chold,
                    float         *XB,
                    int            ldXB,
                    csipl_length   p);
```

Parameters

- chold, structure, input. A Cholesky decomposition object for the matrix A.
- XB, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

Solve the linear system $AX = B$ for a symmetric positive definite matrix A using the decomposition computed by csipl_chold_f.

Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after csipl_chold_f fails. This will result in a non-zero unsuccessful return value.

csipl_ccholsol_inter_f

Solve a Hermitian positive definite linear system.

Prototype

```
int csipl_ccholsol_inter_f(
                    csipl_cchol_f  *chold,
                    void           *XB,
                    int             ldXB,
                    csipl_length    p);
```

Parameters

- chold, structure, input. A Cholesky decomposition object for the matrix A.
- XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

Solve the linear system $AX = B$ for a Hermitian positive definite matrix A using the decomposition computed by csipl_cchold_f.

Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after csipl_cchold_f fails. This will result in a non-zero unsuccessful return value.

csipl_ccholsol_split_f

Solve a Hermitian positive definite linear system.

Prototype

```
int csipl_ccholsol_split_f(
                    csipl_cchol_f  *chold,
                    float          *XB_re,
                    float          *XB_im,
                    int             ldXB,
                    csipl_length    p);
```

Parameters

- chold, structure, input. A Cholesky decomposition object for the matrix A.
- XB_re, real part of complex matrix, size n by p, modified in place.
- XB_im, imaginary part of complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

Solve the linear system $AX = B$ for a Hermitian positive definite matrix A using the decomposition computed by csipl_cchold_f.

Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after csipl_cchold_f fails. This will result in a non-zero unsuccessful return value.

8.5 Overdetermined Linear System Solver

- csipl_qrd_f
- csipl_cqrd_inter_f
- csipl_cqrd_split_f
- csipl_qrd_create_f
- csipl_cqrd_create_f
- csipl_Dqrd_destroy_P
- csipl_Dqrd_getattr_P
- csipl_qrdprodq_f
- csipl_cqrdprodq_inter_f
- csipl_cqrdprodq_split_f
- csipl_qrdsolr_f
- csipl_cqrdsolr_inter_f
- csipl_cqrdsolr_split_f
- csipl_qrsol_f
- csipl_cqrsol_inter_f
- csipl_cqrsol_split_f

csipl_qrd_f

Compute a QR decomposition of a matrix .

Prototype

```
int csipl_qrd_f(
                    csipl_cqr_f *qrd,
                    float       *A,
                    int          ldA);
```

Parameters

- qrd, structure, input.
- A, real matrix, size m by n, modified in place.
- ldA, integer scalar, input.

Return Value

- Error code.

Description

Computes the QR decomposition of an m by n matrix A where $n \leqslant m$.

The QR decomposition of A is given by $A = QR$ where Q is an m by n orthogonal matrix ($Q^{\mathrm{T}}Q = I$) and R is an n by n upper triangular matrix. If A has full rank then R is non-singular. The routine does not perform any column interchanges.

Returns zero on success. It will fail if A does not have full column rank.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero diagonal element of R was encountered.

csipl_cqrd_inter_f

Compute a QR decomposition of a matrix .

Prototype

```
int csipl_cqrd_inter_f(
                    csipl_cqr_f *qrd,
                    void        *A,
                    int          ldA);
```

Parameters

- qrd, structure, input.
- A, complex matrix, size m by n, modified in place.
- ldA, integer scalar, input.

Return Value

- Error code.

Description

Computes the QR decomposition of an m by n matrix A where $n \leqslant m$.

The QR decomposition of A is given by $A = QR$ where Q is an m by n unitary matrix ($Q^{\mathrm{H}}Q = I$) and R is an n by n upper triangular matrix. If A has full rank then R is non-singular. The routine does not perform any column interchanges.

Returns zero on success. It will fail if A does not have full column rank.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero diagonal element of R was encountered.

csipl_cqrd_split_f

Compute a QR decomposition of a matrix .

Prototype

```
int csipl_cqrd_split_f(
                    csipl_cqr_f *qrd,
                    float       *A_re,
                    float       *A_im,
                    int          ldA);
```

Parameters

- qrd, structure, input.
- A_re, real part of complex matrix, size m by n, modified in place.
- A_im, imaginary part of complex matrix, size m by n, modified in place.
- ldA, integer scalar, input.

Return Value

- Error code.

Description

Computes the QR decomposition of an m by n matrix A where $n \leqslant m$.

The QR decomposition of A is given by $A = QR$ where Q is an m by n unitary matrix ($Q^{\mathrm{H}}Q = I$) and R is an n by n upper triangular matrix. If A has full rank then R is non-singular. The routine does not perform any column interchanges.

Returns zero on success. It will fail if A does not have full column rank.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero diagonal element of R was

encountered.

csipl_qrd_create_f

Create a QR decomposition object.

Prototype

```
csipl_qr_f * csipl_qrd_create_f(
                    unsigned int   m,
                    unsigned int   n,
                    csipl_qrd_qopt qopt);
```

Parameters

- m, integer scalar, input.
- n, integer scalar, input.
- qopt, enumerated type, input.

CSIPL_QRD_NOSAVEQ	do not save Q
CSIPL_QRD_SAVEQ	save full Q
CSIPL_QRD_SAVEQ1	save skinny Q

Return Value

- structure.

Description

Creates a QR decomposition object. The QR decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.

The QR decomposition of A is given by $A = QR$ where Q is an m by n orthogonal matrix ($Q^{\mathrm{T}}Q = I$) and R is an n by n upper triangular matrix, and $n \leqslant m$. If A has full rank then R is non-singular.

The matrix R will be generated and retained for later use. There is a flag to indicate whether Q is retained and, if so, in what format.

NULL is returned if the create fails.

Restrictions

Errors

The input arguments must conform to the following:

1. m and n positive with n $\leqslant$ m

2. `qopt` is valid.

Notes

csipl_cqrd_create_f

Create a QR decomposition object.

Prototype

```
csipl_cqr_f * csipl_cqrd_create_f(
                    unsigned int   m,
                    unsigned int   n,
                    csipl_qrd_qopt qopt);
```

Parameters

- m, integer scalar, input.
- n, integer scalar, input.
- qopt, enumerated type, input.

CSIPL_QRD_NOSAVEQ	do not save Q
CSIPL_QRD_SAVEQ	save full Q
CSIPL_QRD_SAVEQ1	save skinny Q

Return Value

- structure.

Description

Creates a QR decomposition object. The QR decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.

The QR decomposition of A is given by $A = QR$ where Q is an m by n unitary matrix ($Q^{\mathrm{H}}Q = I$) and R is an n by n upper triangular matrix, and $n \leqslant m$. If A has full rank then R is non-singular.

The matrix R will be generated and retained for later use. There is a flag to indicate whether Q is retained and, if so, in what format.

NULL is returned if the create fails.

Restrictions

Errors

The input arguments must conform to the following:

1. m and n positive with n $\leqslant$ m

2. `qopt` is valid.

Notes

csipl_Dqrd_destroy_P

Destroy a QR decomposition object.

Prototype

```
int csipl_Dqrd_destroy_P(
                    csipl_Dqr_P *qrd);
```

The following instances are supported:

```
csipl_qrd_destroy_f
csipl_cqrd_destroy_f
```

Parameters

- `qrd`, structure, input.

Return Value

- Error code.

Description

Destroys (frees the memory used by) a QR decomposition object. Returns zero on success, non-zero on failure.

Restrictions

Errors

The input object must conform to the following:

1. The QR decomposition object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

csipl_Dqrd_getattr_P

Returns the attributes of a QR decomposition object.

Prototype

```
void csipl_Dqrd_getattr_P(
                    csipl_Dqr_P       *qrd,
                    csipl_Dqr_attr_P  *attr);
```

The following instances are supported:

csipl_qrd_getattr_f

csipl_cqrd_getattr_f

Parameters

- qrd, structure, input.
- attr, pointer to structure, output.
  The attribute structure contains the following information:

csipl_length	m	number of rows in input matrix
csipl_length	n	number of columns in input matrix
csipl_qrd_qopt	Qopt	matrix Q is saved/not saved

Return Value

- none.

Description

Returns the attributes of a QR decomposition object.

Restrictions

Errors

Notes

csipl_qrdprodq_f

Multiply a matrix by the matrix Q from a QR decomposition.

Prototype

```
int csipl_qrdprodq_f(
                    csipl_qr_f      *qrd,
                    csipl_mat_op     opQ,
                    csipl_mat_side   apQ,
                    float           *C,
                    int              ldC,
                    csipl_length     p);
```

Parameters

- `qrd`, structure, input.
- `opQ`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose

- `apQ`, enumerated type, input.

`CSIPL_MAT_LSIDE`	left side
`CSIPL_MAT_RSIDE`	right side

- `C`, real matrix, size p by q, modified in place.
- `ldC`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- Error code.

Description

This function overwrites a matrix C with the product $\mathrm{op}(Q) \cdot C$ if multiplied on the left, or $C \cdot \mathrm{op}(Q)$ if multiplied on the right. The matrix Q is computed by `csipl_qrd_f` (its size depends on which option was used to store it), and op() is either the identity or transpose operation.

In some cases, the output matrix is larger than C.

Returns zero on success, non-zero on failure.

Restrictions

Since the output data space may be larger than the input data space, it is required that the input allows storage in the block for the output data. This means the row stride and column stride must be calculated to accommodate the larger data space, whether it be input or output.

Errors

The arguments must conform to the following:

1. `opQ` is valid
2. `apQ` is valid
3. The matrix `C` and the QR decomposition object `qrd` must be conformant.
4. The QR decomposition object must have specified retaining the Q matrix when it was created.

Notes

It is safe to call this function after `csipl_qrd_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_cqrdprodq_inter_f

Multiply a matrix by the matrix Q from a QR decomposition.

Prototype

```
int csipl_cqrdprodq_inter_f(
                    csipl_cqr_f     *qrd,
                    csipl_mat_op     opQ,
                    csipl_mat_side   apQ,
                    void            *C,
                    int              ldC,
                    csipl_length     p);
```

Parameters

- `qrd`, structure, input.
- `opQ`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose
`CSIPL_MAT_HERM`	Hermitian (conjugate transpose)

- `apQ`, enumerated type, input.

`CSIPL_MAT_LSIDE`	left side
`CSIPL_MAT_RSIDE`	right side

- `C`, complex matrix, size p by q, modified in place.
- `ldC`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- Error code.

Description

This function overwrites a matrix C with the product $\mathrm{op}(Q) \cdot C$ if multiplied on the left, or $C \cdot \mathrm{op}(Q)$ if multiplied on the right. The matrix Q is computed by `csipl_cqrd_f` (its size depends on which option was used to store it), and op() is either the identity or (identity, transpose, or Hermitian). operation.

In some cases, the output matrix is larger than C.

Returns zero on success, non-zero on failure.

Restrictions

Since the output data space may be larger than the input data space, it is required that the input allows storage in the block for the output data. This means the row stride and column stride must be calculated to accommodate the larger data space, whether it be input or output.

Errors

The arguments must conform to the following:

1. `opQ` is valid
2. `apQ` is valid
3. The matrix `C` and the QR decomposition object `qrd` must be conformant.
4. The QR decomposition object must have specified retaining the Q matrix when it was created.

Notes

It is safe to call this function after `csipl_cqrd_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_cqrdprodq_split_f

Multiply a matrix by the matrix Q from a QR decomposition.

Prototype

```
int csipl_cqrdprodq_split_f(
                    csipl_cqr_f     *qrd_re,
                    csipl_cqr_f     *qrd_im,
                    csipl_mat_op     opQ,
                    csipl_mat_side   apQ,
                    float           *C_re,
                    float           *C_im,
                    int              ldC,
                    csipl_length     p);
```

Parameters

- qrd_re, real part of structure, input.
- qrd_im, imaginary part of structure, input.
- opQ, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose
CSIPL_MAT_HERM	Hermitian (conjugate transpose)

- apQ, enumerated type, input.

CSIPL_MAT_LSIDE	left side
CSIPL_MAT_RSIDE	right side

- C_re, real part of complex matrix, size p by q, modified in place.
- C_im, imaginary part of complex matrix, size p by q, modified in place.
- ldC, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

This function overwrites a matrix C with the product $\mathrm{op}(Q) \cdot C$ if multiplied on the left, or $C \cdot \mathrm{op}(Q)$ if multiplied on the right. The matrix Q is computed by csipl_cqrd_f (its size depends on which option was used to store it), and op() is either the identity or (identity, transpose, or Hermitian). operation.

In some cases, the output matrix is larger than C.

Returns zero on success, non-zero on failure.

Restrictions

Since the output data space may be larger than the input data space, it is required that the input allows storage in the block for the output data. This means the row stride and column stride must be calculated to accommodate the larger data space, whether it be input or output.

Errors

The arguments must conform to the following:

1. `opQ` is valid
2. `apQ` is valid
3. The matrix `C` and the QR decomposition object `qrd` must be conformant.
4. The QR decomposition object must have specified retaining the Q matrix when it was created.

Notes

It is safe to call this function after `csipl_cqrd_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_qrdsolr_f

Solve linear system based on the matrix R, from QR decomposition of the matrix A.

Prototype

```
int csipl_qrdsolr_f(
                    csipl_qr_f    *qrd,
                    csipl_mat_op   OpR,
                    float          alpha,
                    float         *XB,
                    int            ldXB,
                    csipl_length   p);
```

Parameters

- `qrd`, structure, input. A QR decomposition object for the matrix A.
- `OpR`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose

- `alpha`, real scalar, input.
- `XB`, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- `ldXB`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- Error code.

Description

Solves a triangular linear system of the form $\mathrm{op}(R)X = \alpha B$ where op() is either the identity or transpose operation, using the decomposition computed by `csipl_qrd_f`.

R is an n by n upper triangular matrix; X and B are n by p matrices.

Returns zero on success, non-zero on failure.

Restrictions

Errors

The arguments must conform to the following:

1. All of the objects must be valid.
2. `OpR` is valid.
3. The matrix `XB` and the QR decomposition object `qrd` must be conformant.

Notes

It is safe to call this function after `csipl_qrd_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_cqrdsolr_inter_f

Solve linear system based on the matrix R, from QR decomposition of the matrix A.

Prototype

```
int csipl_cqrdsolr_inter_f(
                    csipl_cqr_f      *qrd,
                    csipl_mat_op      OpR,
                    csipl_cscalar_f   alpha,
                    void             *XB,
                    int               ldXB,
                    csipl_length      p);
```

Parameters

- `qrd`, structure, input. A QR decomposition object for the matrix A.
- `OpR`, enumerated type, input.

`CSIPL_MAT_NTRANS`	no transformation
`CSIPL_MAT_TRANS`	transpose
`CSIPL_MAT_HERM`	Hermitian (conjugate transpose)

- `alpha`, complex scalar, input.
- `XB`, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- `ldXB`, integer scalar, input.
- `p`, integer scalar, input.

Return Value

- Error code.

Description

Solves a triangular linear system of the form $\mathrm{op}(R)X = \alpha B$ where op() is either the identity or (identity, transpose, or Hermitian). operation, using the decomposition computed by `csipl_cqrd_f`.

R is an n by n upper triangular matrix; X and B are n by p matrices.

Returns zero on success, non-zero on failure.

Restrictions

Errors

The arguments must conform to the following:

1. All of the objects must be valid.
2. `OpR` is valid.
3. The matrix `XB` and the QR decomposition object `qrd` must be conformant.

Notes

It is safe to call this function after `csipl_cqrd_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_cqrdsolr_split_f

Solve linear system based on the matrix R, from QR decomposition of the matrix A.

Prototype

```
int csipl_cqrdsolr_split_f(
                    csipl_cqr_f      *qrd_re,
                    csipl_cqr_f      *qrd_im,
                    csipl_mat_op      OpR,
                    csipl_cscalar_f   alpha_re,
                    csipl_cscalar_f   alpha_im,
                    float            *XB_re,
                    float            *XB_im,
                    int               ldXB,
                    csipl_length      p);
```

Parameters

- qrd_re, real part of structure, input.
- qrd_im, imaginary part of structure, input. A QR decomposition object for the matrix A.
- OpR, enumerated type, input.

CSIPL_MAT_NTRANS	no transformation
CSIPL_MAT_TRANS	transpose
CSIPL_MAT_HERM	Hermitian (conjugate transpose)

- alpha_re, real part of complex scalar, input.
- alpha_im, imaginary part of complex scalar, input.
- XB_re, real part of complex matrix, size n by p, modified in place.
- XB_im, imaginary part of complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

Solves a triangular linear system of the form $\mathrm{op}(R)X = \alpha B$ where op() is either the identity or (identity, transpose, or Hermitian). operation, using the decomposition computed

by `csipl_cqrd_f`.

R is an n by n upper triangular matrix; X and B are n by p matrices.

Returns zero on success, non-zero on failure.

Restrictions

Errors

The arguments must conform to the following:

1. All of the objects must be valid.
2. `OpR` is valid.
3. The matrix `XB` and the QR decomposition object `qrd` must be conformant.

Notes

It is safe to call this function after `csipl_cqrd_f`. fails. This will result in a non-zero unsuccessful return value.

csipl_qrsol_f

Solve either a linear covariance or linear least squares problem.

Prototype

```
int csipl_qrsol_f(
                    csipl_qr_f      *qrd,
                    csipl_qrd_prob   prob,
                    float           *XB,
                    int              ldXB,
                    csipl_length     p);
```

Parameters

- qrd, structure, input. A QR decomposition object for the matrix A.
- prob, enumerated type, input.

  CSIPL_COV solve a covariance linear system problem
  CSIPL_LLS solve a linear least squares problem
- XB, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

Let A be an m by n matrix of rank n with $n \leqslant m$, and let B be a matrix of size n by p for a covariance problem or m by p for a least squares problem.

This routine solves one of the following problems using the decomposition computed by csipl_qrd_f

- a covariance linear system problem $A^{\mathrm{T}}AX = B$
- a linear least squares problem $\min|AX - B|_2$.

Restrictions

This routine will fail if A does not have full rank.

Errors

The arguments must conform to the following:

1. The matrix `XB` and the QR decomposition object `qrd` must be conformant.
2. `prob` is valid.

Notes

It is safe to call this function after `csipl_qrd_f` fails. This will result in a non-zero unsuccessful return value.

csipl_cqrsol_inter_f

Solve either a linear covariance or linear least squares problem.

Prototype

```
int csipl_cqrsol_inter_f(
                    csipl_cqr_f     *qrd,
                    csipl_qrd_prob   prob,
                    void            *XB,
                    int              ldXB,
                    csipl_length     p);
```

Parameters

- qrd, structure, input. A QR decomposition object for the matrix A.
- prob, enumerated type, input.

  CSIPL_COV solve a covariance linear system problem
  CSIPL_LLS solve a linear least squares problem
- XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

Let A be an m by n matrix of rank n with $n \leqslant m$, and let B be a matrix of size n by p for a covariance problem or m by p for a least squares problem.

This routine solves one of the following problems using the decomposition computed by csipl_qrd_f

- a covariance linear system problem $A^{\mathrm{H}}AX = B$
- a linear least squares problem $\min|AX - B|_2$.

Restrictions

This routine will fail if A does not have full rank.

Errors

The arguments must conform to the following:

1. The matrix `XB` and the QR decomposition object `qrd` must be conformant.
2. `prob` is valid.

Notes

It is safe to call this function after `csipl_cqrd_f` fails. This will result in a non-zero unsuccessful return value.

csipl_cqrsol_split_f

Solve either a linear covariance or linear least squares problem.

Prototype

```
int csipl_cqrsol_split_f(
                    csipl_cqr_f     *qrd_re,
                    csipl_cqr_f     *qrd_im,
                    csipl_qrd_prob   prob,
                    void            *XB,
                    int              ldXB,
                    csipl_length     p);
```

Parameters

- qrd_re, real part of structure, input.
- qrd_im, imaginary part of structure, input. A QR decomposition object for the matrix A.
- prob, enumerated type, input.

  CSIPL_COV solve a covariance linear system problem
  CSIPL_LLS solve a linear least squares problem
- XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.
- ldXB, integer scalar, input.
- p, integer scalar, input.

Return Value

- Error code.

Description

Let A be an m by n matrix of rank n with $n \leqslant m$, and let B be a matrix of size n by p for a covariance problem or m by p for a least squares problem.

This routine solves one of the following problems using the decomposition computed by csipl_qrd_f

- a covariance linear system problem $A^{\mathrm{H}}AX = B$
- a linear least squares problem $\min|AX - B|_2$.

Restrictions

This routine will fail if A does not have full rank.

Errors

The arguments must conform to the following:

1. The matrix `XB` and the QR decomposition object `qrd` must be conformant.
2. `prob` is valid.

Notes

It is safe to call this function after `csipl_cqrd_f` fails. This will result in a non-zero unsuccessful return value.