



NASoftware Limited
Incorporating InfoSAR

CSIPL Quick Reference Guide

CSIPL/Brief [3.1D]

Release 3.1D
May 2013

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	3
1.7	Hints	3
1.8	Notation	3
1.9	Errors and Restrictions	4
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	9
2.12	Linear Algebra Functions	9
2.13	Matrix and Vector Operations	9
2.14	LU Decomposition, Cholesky and QRD Functions	9
2.15	Special Linear System Solvers	10
2.16	Controlling the Number of Threads	10
3	Support Functions	11
3.1	Initialisation and Finalisation	11
	<code>csipl_init</code>	11
	<code>csipl_finalize</code>	11
3.2	Sundry Functions	11
	<code>csipl_complete</code>	11

	csipl_cstorage	11
4	Scalar Functions	12
4.1	Real Scalar Functions	12
	csipl_acos_P	12
	csipl_asin_P	12
	csipl_atan_P	12
	csipl_atan2_P	12
	csipl_ceil_P	12
	csipl_cos_P	12
	csipl_cosh_P	12
	csipl_exp_P	12
	csipl_floor_P	13
	csipl_log_P	13
	csipl_log10_P	13
	csipl_mag_P	13
	csipl_pow_P	13
	csipl_sin_P	13
	csipl_sinh_P	13
	csipl_sqrt_P	13
	csipl_tan_P	13
	csipl_tanh_P	14
4.2	Complex Scalar Functions	14
	csipl_arg_P	14
	csipl_cadd_P	14
	csipl_rcadd_P	14
	csipl_cdiv_P	14
	csipl_crdiv_P	14
	csipl_cexp_P	14
	csipl_cjmul_P	14
	csipl_cmag_P	15
	csipl_cmagsq_P	15
	csipl_cmplx_P	15
	csipl_cmul_P	15
	csipl_rcmul_P	15
	csipl_cneg_P	15
	csipl_conj_P	15
	csipl_crecip_P	15

	csipl_csqrt_P	15
	csipl_csub_P	16
	csipl_rsub_P	16
	csipl_csub_P	16
	csipl_imag_P	16
	csipl_polar_P	16
	csipl_real_P	16
	csipl_rect_P	16
4.3	Index Scalar Functions	16
	csipl_matindex	16
	csipl_mcolindex	16
	csipl_mrowindex	17
5	Random Number Generation	18
5.1	Random Number Functions	18
	csipl_randcreate	18
	csipl_randdestroy	18
	csipl_randu_P	18
	csipl_crandu_P	18
	csipl_vrandu_P	18
	csipl_cvrandu_inter_P	19
	csipl_cvrandu_split_P	19
	csipl_randn_P	19
	csipl_crandn_P	19
	csipl_vrandn_P	19
	csipl_cvrandn_inter_P	20
	csipl_cvrandn_split_P	20
6	Vector And Elementwise Operations	21
6.1	Elementary Mathematical Functions	21
	csipl_vacos_P	21
	csipl_macos_P	21
	csipl_vasin_P	21
	csipl_masin_P	21
	csipl_vatan_P	21
	csipl_matan_P	21
	csipl_vatan2_P	22
	csipl_matan2_P	22
	csipl_vcos_P	22

csipl_mcos_P	22
csipl_vcosh_P	22
csipl_mcosh_P	22
csipl_vexp_P	22
csipl_cvexp_inter_P	23
csipl_cvexp_split_P	23
csipl_mexp_P	23
csipl_cmexp_inter_P	23
csipl_cmexp_split_P	23
csipl_vexp10_P	23
csipl_mexp10_P	23
csipl_vfloor_P	24
csipl_vlog_P	24
csipl_cvlog_inter_P	24
csipl_cvlog_split_P	24
csipl_mlog_P	24
csipl_cmlog_inter_P	24
csipl_cmlog_split_P	24
csipl_vlog10_P	25
csipl_mlog10_P	25
csipl_vsin_P	25
csipl_msin_P	25
csipl_vsinh_P	25
csipl_msinh_P	25
csipl_vsqrt_P	25
csipl_cvsqrt_inter_P	25
csipl_cvsqrt_split_P	26
csipl_msqrt_P	26
csipl_cmsqrt_inter_P	26
csipl_cmsqrt_split_P	26
csipl_mtanh_P	26
csipl_mtanh_P	26
6.2 Unary Operations	26
csipl_varg_P	27
csipl_marg_P	27
csipl_vceil_P	27
csipl_cvconj_inter_P	27
csipl_cvconj_split_P	27

csipl_cmconj_inter_P	27
csipl_cmconj_split_P	27
csipl_vcumsum_P	28
csipl_cvcumsum_inter_P	28
csipl_cvcumsum_split_P	28
csipl_mcumsum_P	28
csipl_cmcumsum_inter_P	28
csipl_cmcumsum_split_P	28
csipl_veuler_P	28
csipl_meuler_P	29
csipl_vmag_P	29
csipl_cvmag_inter_P	29
csipl_cvmag_split_P	29
csipl_mmag_P	29
csipl_cmmag_P	29
csipl_cmmag_split_P	29
csipl_vcmagsq_inter_P	30
csipl_vcmagsq_split_P	30
csipl_mcmagsq_P	30
csipl_vmeanval_P	30
csipl_cvmeanval_inter_P	30
csipl_cvmeanval_split_P	30
csipl_mmeanval_P	30
csipl_cmmeanval_inter_P	30
csipl_cmmeanval_split_P	30
csipl_vmeansqval_P	31
csipl_cvmeansqval_inter_P	31
csipl_cvmeansqval_split_P	31
csipl_mmeansqval_P	31
csipl_cmmeansqval_inter_P	31
csipl_cmmeansqval_split_P	31
csipl_vmodulate_P	31
csipl_cvmodulate_inter_P	31
csipl_cvmodulate_split_P	32
csipl_vneg_P	32
csipl_cvneg_inter_P	32
csipl_cvneg_split_P	32
csipl_mneg_P	32

csipl_cmneg_inter_P	32
csipl_cmneg_split_P	32
csipl_vrecip_P	33
csipl_cvrecip_inter_P	33
csipl_cvrecip_split_P	33
csipl_mrecip_P	33
csipl_cmrecip_inter_P	33
csipl_cmrecip_split_P	33
csipl_vrsqrt_P	33
csipl_mrsqrt_P	34
csipl_vsqr_P	34
csipl_msqr_P	34
csipl_cvsumval_inter_P	34
csipl_cvsumval_split_P	34
csipl_msumval_P	34
csipl_cmsumval_inter_P	34
csipl_cmsumval_split_P	34
csipl_msumsqval_P	35
6.3 Binary Operations	35
csipl_vadd_P	35
csipl_cvadd_inter_P	35
csipl_cvadd_split_P	35
csipl_madd_P	35
csipl_cmadd_inter_P	35
csipl_cmadd_split_P	36
csipl_rcvadd_P	36
csipl_rcvadd_split_P	36
csipl_rcmadd_inter_P	36
csipl_rcmadd_split_P	36
csipl_svadd_P	37
csipl_csvadd_inter_P	37
csipl_csvadd_split_P	37
csipl_smadd_P	37
csipl_csmadd_inter_P	37
csipl_csmadd_split_P	37
csipl_rscvadd_inter_P	38
csipl_rscvadd_split_P	38
csipl_rscmadd_inter_P	38

csipl_rscmadd_split_P	38
csipl_Dvdiv_P	38
csipl_Dvdiv_inter_P	38
csipl_Dvdiv_split_P	39
csipl_mdiv_P	39
csipl_cmdiv_inter_P	39
csipl_cmdiv_split_P	39
csipl_rcvdiv_inter_P	39
csipl_rcvdiv_split_P	39
csipl_rcmdiv_inter_P	40
csipl_rcmdiv_split_P	40
csipl_crmdiv_inter_P	40
csipl_crmdiv_split_P	40
csipl_rscmsub_inter_P	40
csipl_rscmsub_split_P	40
csipl_vsdiv_P	41
csipl_cvrsdiv_inter_P	41
csipl_cvrsdiv_split_P	41
csipl_msdiv_P	41
csipl_cmrsdiv_inter_P	41
csipl_cmrsdiv_split_P	41
csipl_vexpoavg_P	42
csipl_cvexpoavg_inter_P	42
csipl_cvexpoavg_split_P	42
csipl_mexpoavg_P	42
csipl_cmexpoavg_inter_P	42
csipl_cmexpoavg_split_P	42
csipl_vhypot_P	43
csipl_mhypot_P	43
csipl_cvjmul_inter_P	43
csipl_cvjmul_split_P	43
csipl_cmjmul_inter_P	43
csipl_cmjmul_split_P	43
csipl_Dvmul_P	44
csipl_cvmul_inter_P	44
csipl_cvmul_split_P	44
csipl_mmul_P	44
csipl_cmmul_inter_P	44

csipl_cmmul_split_P	45
csipl_rcvmul_inter_P	45
csipl_rcvmul_split_P	45
csipl_rcmmul_inter_P	45
csipl_rcmmul_split_P	45
csipl_rscvmul_inter_P	45
csipl_rscvmul_split_P	46
csipl_csvm_inter_P	46
csipl_csvm_split_P	46
csipl_smmul_P	46
csipl_csmmul_inter_P	46
csipl_csmmul_split_P	46
csipl_rscmmul_inter_P	47
csipl_rscmmul_split_P	47
csipl_vmmul_P	47
csipl_cvmmul_inter_P	47
csipl_cvmmul_split_P	47
csipl_rvcmmul_inter_P	48
csipl_rvcmmul_split_P	48
csipl_vsub_P	48
csipl_cvsub_inter_P	48
csipl_cvsub_split_P	48
csipl_msub_P	49
csipl_cmsub_inter_P	49
csipl_cmsub_split_P	49
csipl_crmsub_inter_P	49
csipl_crmsub_split_P	49
csipl_rcvsub_inter_P	50
csipl_rcvsub_split_P	50
csipl_rcmsub_inter_P	50
csipl_rcmsub_split_P	50
csipl_crvsub_inter_P	50
csipl_crvsub_split_P	50
csipl_svsub_P	51
csipl_csvsub_inter_P	51
csipl_csvsub_split_P	51
csipl_smsub_P	51
csipl_csmsub_inter_P	51

	csipl_csmsub_split_P	51
	csipl_smdiv_P	52
	csipl_csmdiv_inter_P	52
	csipl_csmdiv_split_P	52
	csipl_rscvdiv_inter_P	52
	csipl_rscvdiv_split_P	52
	csipl_rscvsub_inter_P	52
	csipl_rscvsub_split_P	53
	csipl_rscmdiv_inter_P	53
	csipl_rscmdiv_split_P	53
6.4	Ternary Operations	53
	csipl_vam_P	53
	csipl_cvam_inter_P	53
	csipl_cvam_split_P	54
	csipl_vmsa_P	54
	csipl_cvmsa_inter_P	54
	csipl_cvmsa_split_P	54
	csipl_vmsb_P	54
	csipl_cvmsb_inter_P	55
	csipl_cvmsb_split_P	55
	csipl_vsam_P	55
	csipl_cvsam_inter_P	55
	csipl_cvsam_split_P	55
	csipl_vsbm_P	56
	csipl_cvsbm_inter_P	56
	csipl_cvsbm_split_P	56
	csipl_vsma_P	56
	csipl_cvsma_inter_P	56
	csipl_cvsma_split_P	57
	csipl_vsmsa_P	57
	csipl_cvmsa_inter_P	57
	csipl_cvmsa_split_P	57
6.5	Logical Operations	57
	csipl_valltrue_bl	57
	csipl_malltrue_bl	57
	csipl_vanytrue_bl	58
	csipl_manytrue_bl	58
	csipl_vleq_P	58

csipl_cvleq_inter_P	58
csipl_cvleq_split_P	58
csipl_mleq_P	58
csipl_cmleq_inter_P	58
csipl_cmleq_split_P	59
csipl_vlge_P	59
csipl_mlge_P	59
csipl_vlgt_P	59
csipl_mlgt_P	59
csipl_vlle_P	59
csipl_mlle_P	60
csipl_vllt_P	60
csipl_mllt_P	60
csipl_vlne_P	60
csipl_cvlne_inter_P	60
csipl_cvlne_split_P	60
csipl_mlne_P	61
csipl_cmlne_inter_P	61
csipl_cmlne_split_P	61
6.6 Selection Operations	61
csipl_vclip_P	61
csipl_vinvclip_P	61
csipl_vindexbool	62
csipl_vmax_P	62
csipl_vmaxmg_P	62
csipl_vcmaxmgsq_inter_P	62
csipl_vcmaxmgsq_split_P	62
csipl_vcmaxmgsqval_inter_P	62
csipl_vcmaxmgsqval_split_P	63
csipl_vmaxmgval_P	63
csipl_vmaxval_P	63
csipl_vmin_P	63
csipl_vminmg_P	63
csipl_vcminmgsq_inter_P	63
csipl_vcminmgsq_split_P	64
csipl_vcminmgsqval_inter_P	64
csipl_vcminmgsqval_split_P	64
csipl_vminmgval_P	64

	csipl_vminval_P	64
6.7	Bitwise and Boolean Logical Operators	64
	csipl_vand_P	64
	csipl_mand_P	65
	csipl_vnot_P	65
	csipl_mnot_P	65
	csipl_vor_P	65
	csipl_mor_P	65
	csipl_mxor_P	65
6.8	Element Generation and Copy	65
	csipl_vcopy_P_P	66
	csipl_cvcopy_inter_P_P	66
	csipl_cvcopy_split_P_P	66
	csipl_mcopy_P_P	67
	csipl_cmcopy_inter_P_P	67
	csipl_cmcopy_split_P_P	67
	csipl_vfill_P	67
	csipl_cvfill_inter_P	67
	csipl_cvfill_split_P	68
	csipl_mfill_P	68
	csipl_cmfill_inter_P	68
	csipl_cmfill_split_P	68
	csipl_vramp_P	68
6.9	Manipulation Operations	68
	csipl_vcplx_inter_P	68
	csipl_vcplx_split_P	69
	csipl_mcplx_P	69
	csipl_vgather_P	69
	csipl_cvgather_inter_P	69
	csipl_cvgather_split_P	69
	csipl_mgather_P	70
	csipl_cmgather_inter_P	70
	csipl_cmgather_split_P	70
	csipl_vimag_inter_P	70
	csipl_vimag_split_P	70
	csipl_mimag_P	70
	csipl_vpolar_inter_P	71
	csipl_vpolar_split_P	71

csipl_mpolar_P	71
csipl_vreal_inter_P	71
csipl_vreal_split_P	71
csipl_mreal_P	71
csipl_vrect_inter_P	72
csipl_vrect_split_P	72
csipl_mrect_P	72
csipl_vscatter_P	72
csipl_cvscatter_inter_P	72
csipl_cvscatter_split_P	73
csipl_mscatter_P	73
csipl_cmscatter_inter_P	73
csipl_cmscatter_split_P	73
csipl_cvswap_inter_P	73
csipl_cvswap_split_P	74
csipl_mswap_P	74
csipl_cmswap_inter_P	74
csipl_cmswap_split_P	74

7 Signal Processing Functions 75

7.1 FFT Functions	75
csipl_ccfftip_create_P	75
csipl_ccfftop_create_P	75
csipl_crfftop_create_P	75
csipl_rcfftop_create_P	75
csipl_ccfftip_inter_P	75
csipl_ccfftip_split_P	75
csipl_ccfftop_inter_P	75
csipl_ccfftop_split_P	76
csipl_crfftop_inter_P	76
csipl_crfftop_split_P	76
csipl_rcfftop_inter_P	76
csipl_rcfftop_split_P	76
csipl_fft_destroy_P	76
csipl_fft_getattr_P	76
csipl_ccfftmop_create_P	76
csipl_ccfftmip_create_P	77
csipl_crfftmop_create_P	77

csipl_rcfftmap_create_P	77
csipl_fftm_destroy_P	77
csipl_fftm_getattr_P	77
csipl_ccfftmip_inter_P	77
csipl_ccfftmip_split_P	77
csipl_ccfftmap_inter_P	77
csipl_ccfftmap_split_P	78
csipl_crfftmap_inter_P	78
csipl_crfftmap_split_P	78
csipl_rcfftmap_inter_P	78
csipl_rcfftmap_split_P	78
csipl_ccfft2dop_create_P	78
csipl_ccfft2dip_create_P	78
csipl_crfft2dop_create_P	78
csipl_rcfft2dop_create_P	79
csipl_ccfft2dip_inter_P	79
csipl_ccfft2dip_split_P	79
csipl_ccfft2dop_inter_P	79
csipl_ccfft2dop_split_P	79
csipl_crfft2dop_inter_P	79
csipl_crfft2dop_split_P	79
csipl_rcfft2dop_inter_P	79
csipl_rcfft2dop_split_P	80
csipl_fft2d_destroy_P	80
csipl_fft2d_getattr_P	80
7.2 Convolution/Correlation Functions	80
csipl_conv1d_create_P	80
csipl_conv1d_destroy_P	80
csipl_conv1d_getattr_P	80
csipl_convolve1d_P	80
csipl_conv2d_create_P	81
csipl_conv2d_destroy_P	81
csipl_conv2d_getattr_P	81
csipl_convolve2d_P	81
csipl_Dcorr1d_create_P	81
csipl_Dcorr1d_destroy_P	81
csipl_Dcorr1d_getattr_P	81
csipl_correlate1d_P	82

	csipl_ccorrelate1d_inter_P	82
	csipl_ccorrelate1d_split_P	82
	csipl_Dcorr2d_create_P	82
	csipl_Dcorr2d_destroy_P	82
	csipl_Dcorr2d_getattr_P	82
	csipl_correlate2d_P	83
	csipl_ccorrelate2d_inter_P	83
	csipl_ccorrelate2d_split_P	83
7.3	Window Functions	83
	csipl_vcreate_blackman_P	83
	csipl_vcreate_cheby_P	83
	csipl_vcreate_hanning_P	84
	csipl_vcreate_kaiser_P	84
7.4	Filter Functions	84
	csipl_fir_create_P	84
	csipl_cfir_create_inter_P	84
	csipl_cfir_create_split_P	84
	csipl_Dfir_destroy_P	84
	csipl_firfft_P	85
	csipl_cfirfft_inter_P	85
	csipl_cfirfft_split_P	85
	csipl_Dfir_getattr_P	85
	csipl_Dfir_reset_P	85
7.5	Miscellaneous Signal Processing Functions	86
	csipl_vhisto_P	86
8	Linear Algebra	87
8.1	Matrix and Vector Operations	87
	csipl_cmherm_inter_P	87
	csipl_cmherm_split_P	87
	csipl_cvjdot_inter_P	87
	csipl_cvjdot_split_P	87
	csipl_gemp_P	87
	csipl_cgemp_inter_P	88
	csipl_cgemp_split_P	88
	csipl_gems_P	88
	csipl_cgems_inter_P	88
	csipl_cgems_split_P	89

csipl_mprod_P	89
csipl_cmprod_inter_P	89
csipl_cmprod_split_P	89
csipl_cmprodh_inter_P	89
csipl_cmprodh_split_P	90
csipl_cmprodj_inter_P	90
csipl_cmprodj_split_P	90
csipl_mprodt_P	90
csipl_cmprodt_inter_P	90
csipl_cmprodt_split_P	91
csipl_mvprod_P	91
csipl_cmvprod_inter_P	91
csipl_cmvprod_split_P	91
csipl_mtrans_P	91
csipl_cmtrans_inter_P	92
csipl_cmtrans_split_P	92
csipl_vdot_P	92
csipl_cvdot_inter_P	92
csipl_cvdot_split_P	92
csipl_vmprod_P	92
csipl_cvmprod_inter_P	92
csipl_cvmprod_split_P	93
csipl_vouter_P	93
csipl_cvouter_inter_P	93
csipl_cvouter_split_P	93
csipl_vcsummgval_inter_P	93
csipl_vcsummgval_split_P	93
csipl_minvlu_P	94
csipl_cminvlu_inter_P	94
csipl_cminvlu_split_P	94
8.2 Special Linear System Solvers	95
csipl_covsol_P	95
csipl_ccovsol_inter_P	95
csipl_ccovsol_split_P	95
csipl_llsqsol_P	95
csipl_cllsqsol_inter_P	95
csipl_cllsqsol_split_P	95
csipl_toepsol_P	96

	csipl_ctoepsol_inter_P	96
	csipl_ctoepsol_split_P	96
8.3	General Square Linear System Solver	96
	csipl_lud_P	96
	csipl_clud_inter_P	96
	csipl_clud_split_P	96
	csipl_Dlud_create_P	97
	csipl_Dlud_destroy_P	97
	csipl_Dlud_getattr_P	97
	csipl_lusol_P	97
	csipl_clusol_inter_P	97
	csipl_clusol_split_P	97
8.4	Symmetric Positive Definite Linear System Solver	97
	csipl_chold_P	97
	csipl_cchold_inter_P	98
	csipl_cchold_split_P	98
	csipl_chold_create_P	98
	csipl_cchold_create_P	98
	csipl_Dchold_destroy_P	98
	csipl_Dchold_getattr_P	98
	csipl_cholsol_P	98
	csipl_ccholsol_inter_P	98
	csipl_ccholsol_split_P	99
8.5	Overdetermined Linear System Solver	99
	csipl_qrd_P	99
	csipl_cqrd_inter_P	99
	csipl_cqrd_split_P	99
	csipl_qrd_create_P	99
	csipl_cqrd_create_P	99
	csipl_Dqrd_destroy_P	99
	csipl_Dqrd_getattr_P	99
	csipl_qrdprodq_P	100
	csipl_cqrdprodq_inter_P	100
	csipl_cqrdprodq_split_P	100
	csipl_qrdsolr_P	100
	csipl_cqrdsolr_inter_P	100
	csipl_cqrdsolr_split_P	100
	csipl_qrsol_P	101

csipl_cqrsol_inter_P	101
csipl_cqrsol_split_P	101

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:

<code>csipl_bool</code>	boolean
<code>csipl_cscalar_f</code>	complex floating-point scalar
<code>csipl_cscalar_d</code>	complex double precision scalar
<code>csipl_index</code>	index to a vector
<code>csipl_index_mi</code>	index (pair of integers) to a matrix
<code>csipl_length</code>	dimension of a vector or matrix
<code>csipl_stride</code>	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
d	double
i	integer
bl	boolean
vi	vector index
mi	matrix index.

1.7 Hints

The 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
$ x $	absolute value of the real number x
$ z $	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 $C[j * \text{strideC}] := A[j * \text{strideA}] + B[j * \text{strideB}]$. Often the range of the index variable is not given explicitly; 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 behaviour, 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 Alignment
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	minimum Vector Length (floats)
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 behaviour. 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 vectorised.

2.7 Random Number Generation

The random number generation functions have not been vectorised 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 optimised to minimise 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 optimised to minimise 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 optimised to minimise 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. Utilising 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 initialisation routine `csip_init` is called.

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

Chapter 3. Support Functions

3.1 Initialisation and Finalisation

Prototype	Description
<pre><i>int</i> csipl_init(void * ptr);</pre>	Provides initialisation, allowing the library to allocate and set a global state, and prepare to support the use of CSIPL functionality by the user.
<pre><i>int</i> csipl_finalize(void * ptr);</pre>	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 <code>csipl_init</code> are no longer in use after the call is complete.

3.2 Sundry Functions

Prototype	Description
<pre><i>void</i> csipl_complete(void);</pre>	Force all deferred CSIPL execution to complete.
<pre><i>csipl_cmplx_mem</i> csipl_cstorage(void);</pre>	Returns the preferred complex storage format for the system.

Chapter 4. Scalar Functions

4.1 Real Scalar Functions

Prototype	Description
<pre>scalar_P csipl_acos_P(scalar_P A);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse cosine of a scalar. The following instances are supported: <code>csipl_acos_f</code> <code>csipl_acos_d</code>
<pre>scalar_P csipl_asin_P(scalar_P A);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse sine of a scalar. The following instances are supported: <code>csipl_asin_f</code> <code>csipl_asin_d</code>
<pre>scalar_P csipl_atan_P(scalar_P A);</pre>	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent of a scalar. The following instances are supported: <code>csipl_atan_f</code> <code>csipl_atan_d</code>
<pre>scalar_P csipl_atan2_P(scalar_P A, scalar_P B);</pre>	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of two scalars. The following instances are supported: <code>csipl_atan2_f</code> <code>csipl_atan2_d</code>
<pre>scalar_P csipl_ceil_P(scalar_P A);</pre>	Computes the ceiling of a scalar. The following instances are supported: <code>csipl_ceil_f</code> <code>csipl_ceil_d</code>
<pre>scalar_P csipl_cos_P(scalar_P A);</pre>	Computes the cosine of a scalar angle in radians. The following instances are supported: <code>csipl_cos_f</code> <code>csipl_cos_d</code>
<pre>scalar_P csipl_cosh_P(scalar_P A);</pre>	Computes the hyperbolic cosine of a scalar. The following instances are supported: <code>csipl_cosh_f</code> <code>csipl_cosh_d</code>
<pre>scalar_P csipl_exp_P(scalar_P A);</pre>	Computes the exponential of a scalar. The following instances are supported: <code>csipl_exp_f</code> <code>csipl_exp_d</code>

Prototype	Description
<pre>scalar_P csipl_floor_P(scalar_P A);</pre>	<p>Computes the floor of a scalar. The following instances are supported:</p> <pre>csipl_floor_f csipl_floor_d</pre>
<pre>scalar_P csipl_log_P(scalar_P A);</pre>	<p>Computes the natural logarithm of a scalar. The following instances are supported:</p> <pre>csipl_log_f csipl_log_d</pre>
<pre>scalar_P csipl_log10_P(scalar_P A);</pre>	<p>Computes the base 10 logarithm of a scalar. The following instances are supported:</p> <pre>csipl_log10_f csipl_log10_d</pre>
<pre>scalar_P csipl_mag_P(scalar_P A);</pre>	<p>Computes the magnitude (absolute value) of a scalar. The following instances are supported:</p> <pre>csipl_mag_f csipl_mag_d</pre>
<pre>scalar_P csipl_pow_P(scalar_P A, scalar_P B);</pre>	<p>Computes the power function of two scalars. The following instances are supported:</p> <pre>csipl_pow_f csipl_pow_d</pre>
<pre>scalar_P csipl_sin_P(scalar_P A);</pre>	<p>Computes the sine of a scalar angle in radians. The following instances are supported:</p> <pre>csipl_sin_f csipl_sin_d</pre>
<pre>scalar_P csipl_sinh_P(scalar_P A);</pre>	<p>Computes the hyperbolic sine of a scalar. The following instances are supported:</p> <pre>csipl_sinh_f csipl_sinh_d</pre>
<pre>scalar_P csipl_sqrt_P(scalar_P A);</pre>	<p>Computes the square root of a scalar. The following instances are supported:</p> <pre>csipl_sqrt_f csipl_sqrt_d</pre>
<pre>scalar_P csipl_tan_P(scalar_P A);</pre>	<p>Computes the tangent of a scalar angle in radians. The following instances are supported:</p> <pre>csipl_tan_f csipl_tan_d</pre>

Prototype	Description
<pre>scalar_P csipl_tanh_P(scalar_P A);</pre>	<p>Computes the hyperbolic tangent of a scalar. The following instances are supported:</p> <pre>csipl_tanh_f csipl_tanh_d</pre>

4.2 Complex Scalar Functions

Prototype	Description
<pre>scalar_P csipl_arg_P(csipl_cscalar_P x);</pre>	<p>Returns the argument in radians $[-\pi, \pi]$ of a complex scalar. The following instances are supported:</p> <pre>csipl_arg_f csipl_arg_d</pre>
<pre>csipl_cscalar_P csipl_cadd_P(csipl_cscalar_P x, csipl_cscalar_P y);</pre>	<p>Computes the complex sum of two scalars. The following instances are supported:</p> <pre>csipl_cadd_f csipl_cadd_d</pre>
<pre>csipl_cscalar_P csipl_rcadd_P(scalar_P x, csipl_cscalar_P y);</pre>	<p>Computes the complex sum of two scalars. The following instances are supported:</p> <pre>csipl_rcadd_f csipl_rcadd_d</pre>
<pre>csipl_cscalar_P csipl_cdiv_P(csipl_cscalar_P x, csipl_cscalar_P y);</pre>	<p>Computes the complex quotient of two scalars. The following instances are supported:</p> <pre>csipl_cdiv_f csipl_cdiv_d</pre>
<pre>csipl_cscalar_P csipl_crdiv_P(csipl_cscalar_P x, scalar_P y);</pre>	<p>Computes the complex quotient of two scalars. The following instances are supported:</p> <pre>csipl_crdiv_f csipl_crdiv_d</pre>
<pre>csipl_cscalar_P csipl_cexp_P(csipl_cscalar_P x);</pre>	<p>Computes the complex exponential of a scalar. The following instances are supported:</p> <pre>csipl_cexp_f csipl_cexp_d</pre>
<pre>csipl_cscalar_P csipl_cjmul_P(csipl_cscalar_P x, csipl_cscalar_P y);</pre>	<p>Computes the product a complex scalar with the conjugate of a second complex scalar. The following instances are supported:</p> <pre>csipl_cjmul_f csipl_cjmul_d</pre>

Prototype	Description
<pre>scalar_P csipl_cmag_P(csipl_cscalar_P x);</pre>	<p>Computes the magnitude of a complex scalar. The following instances are supported:</p> <pre>csipl_cmag_f csipl_cmag_d</pre>
<pre>scalar_P csipl_cmagsq_P(csipl_cscalar_P x);</pre>	<p>Computes the magnitude squared of a complex scalar. The following instances are supported:</p> <pre>csipl_cmagsq_f csipl_cmagsq_d</pre>
<pre>csipl_cscalar_P csipl_cmplx_P(scalar_P re, scalar_P im);</pre>	<p>Form a complex scalar from two real scalars. The following instances are supported:</p> <pre>csipl_cmplx_f csipl_cmplx_d</pre>
<pre>csipl_cscalar_P csipl_cmul_P(csipl_cscalar_P x, csipl_cscalar_P y);</pre>	<p>Computes the complex product of two scalars. The following instances are supported:</p> <pre>csipl_cmul_f csipl_cmul_d</pre>
<pre>csipl_cscalar_P csipl_rcmul_P(scalar_P x, csipl_cscalar_P y);</pre>	<p>Computes the complex product of two scalars. The following instances are supported:</p> <pre>csipl_rcmul_f csipl_rcmul_d</pre>
<pre>csipl_cscalar_P csipl_cneg_P(csipl_cscalar_P x);</pre>	<p>Computes the negation of a complex scalar. The following instances are supported:</p> <pre>csipl_cneg_f csipl_cneg_d</pre>
<pre>csipl_cscalar_P csipl_conj_P(csipl_cscalar_P x);</pre>	<p>Computes the complex conjugate of a scalar. The following instances are supported:</p> <pre>csipl_conj_f csipl_conj_d</pre>
<pre>csipl_cscalar_P csipl_crecip_P(csipl_cscalar_P x);</pre>	<p>Computes the reciprocal of a complex scalar. The following instances are supported:</p> <pre>csipl_crecip_f csipl_crecip_d</pre>
<pre>csipl_cscalar_P csipl_csqrt_P(csipl_cscalar_P x);</pre>	<p>Computes the square root a complex scalar. The following instances are supported:</p> <pre>csipl_csqrt_f csipl_csqrt_d</pre>

Prototype	Description
<pre> <i>csipl_cscalar_P</i> <i>csipl_csub_P</i>(<i>csipl_cscalar_P</i> x, <i>csipl_cscalar_P</i> y); </pre>	<p>Computes the complex difference of two scalars. The following instances are supported:</p> <p><i>csipl_csub_f</i> <i>csipl_csub_d</i></p>
<pre> <i>csipl_cscalar_P</i> <i>csipl_rcsub_P</i>(<i>scalar_P</i> x, <i>csipl_cscalar_P</i> y); </pre>	<p>Computes the complex difference of two scalars. The following instances are supported:</p> <p><i>csipl_rcsub_f</i> <i>csipl_rcsub_d</i></p>
<pre> <i>csipl_cscalar_P</i> <i>csipl_crsub_P</i>(<i>csipl_cscalar_P</i> x, <i>scalar_P</i> y); </pre>	<p>Computes the complex difference of two scalars. The following instances are supported:</p> <p><i>csipl_crsub_f</i> <i>csipl_crsub_d</i></p>
<pre> <i>scalar_P</i> <i>csipl_imag_P</i>(<i>csipl_cscalar_P</i> x); </pre>	<p>Extract the imaginary part of a complex scalar. The following instances are supported:</p> <p><i>csipl_imag_f</i> <i>csipl_imag_d</i></p>
<pre> void <i>csipl_polar_P</i>(<i>csipl_cscalar_P</i> a, <i>scalar_P</i> * r, <i>scalar_P</i> * t); </pre>	<p>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. The following instances are supported:</p> <p><i>csipl_polar_f</i> <i>csipl_polar_d</i></p>
<pre> <i>scalar_P</i> <i>csipl_real_P</i>(<i>csipl_cscalar_P</i> x); </pre>	<p>Extract the real part of a complex scalar. The following instances are supported:</p> <p><i>csipl_real_f</i> <i>csipl_real_d</i></p>
<pre> <i>csipl_cscalar_P</i> <i>csipl_rect_P</i>(<i>scalar_P</i> r, <i>scalar_P</i> t); </pre>	<p>Convert a pair of real scalars from complex polar to complex rectangular form. The following instances are supported:</p> <p><i>csipl_rect_f</i> <i>csipl_rect_d</i></p>

4.3 Index Scalar Functions

Prototype	Description
<pre> <i>csipl_scalar_mi</i> <i>csipl_matindex</i>(<i>csipl_index</i> r, <i>csipl_index</i> c); </pre>	<p>Form a matrix index from two vector indices.</p>
<pre> <i>csipl_index</i> <i>csipl_mcolindex</i>(<i>csipl_scalar_mi</i> mi); </pre>	<p>Returns the column vector index from a matrix index.</p>

Prototype	Description
<code>csipl_index</code> <code>csipl_mrowindex</code> (<code>csipl_scalar_mi mi</code>);	Returns the row vector index from a matrix index.

Chapter 5. Random Number Generation

5.1 Random Number Functions

Prototype	Description
<pre><code>csipl_randstate * csipl_randcreate(csipl_index seed, csipl_index numprocs, csipl_index id, csipl_rng portable);</code></pre>	Create a random number generator state object.
<pre><code>int csipl_randdestroy(csipl_randstate * rand);</code></pre>	Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure.
<pre><code>scalar_P csipl_randu_P(csipl_randstate * state);</code></pre>	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 $(0, 2^{31} - 1)$. The following instances are supported: <code>csipl_randu_f</code> <code>csipl_randu_d</code>
<pre><code>csipl_cscalar_P csipl_crandu_P(csipl_randstate * state);</code></pre>	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 $(0, 2^{31} - 1)$. The following instances are supported: <code>csipl_crandu_f</code> <code>csipl_crandu_d</code>
<pre><code>void csipl_vrandu_P(csipl_randstate * state, scalar_P * R, csipl_stride strideR, csipl_length n);</code></pre>	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 $(0, 2^{31} - 1)$. The following instances are supported: <code>csipl_vrandu_f</code> <code>csipl_vrandu_d</code>

Prototype	Description
<pre>void csipl_cvrandu_inter_P(csipl_randstate * state, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>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 $(0, 2^{31} - 1)$.</p> <p>The following instances are supported:</p> <p>csipl_cvrandu_inter_f csipl_cvrandu_inter_d</p>
<pre>void csipl_cvrandu_split_P(csipl_randstate * state, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>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 $(0, 2^{31} - 1)$.</p> <p>The following instances are supported:</p> <p>csipl_cvrandu_split_f csipl_cvrandu_split_d</p>
<pre>scalar_P csipl_randn_P(csipl_randstate * state);</pre>	<p>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. The following instances are supported:</p> <p>csipl_randn_f csipl_randn_d</p>
<pre>csipl_cscalar_P csipl_crandn_P(csipl_randstate * state);</pre>	<p>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. The following instances are supported:</p> <p>csipl_crandn_f csipl_crandn_d</p>
<pre>scalar_P csipl_vrandn_P(csipl_randstate * state);</pre>	<p>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. The following instances are supported:</p> <p>csipl_vrandn_f csipl_vrandn_d</p>

Prototype	Description
<pre>void csipl_cvrandn_inter_P(csipl_randstate * state, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>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. The following instances are supported:</p> <p>csipl_cvrandn_inter_f csipl_cvrandn_inter_d</p>
<pre>void csipl_cvrandn_split_P(csipl_randstate * state, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>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. The following instances are supported:</p> <p>csipl_cvrandn_split_f csipl_cvrandn_split_d</p>

Chapter 6. Vector And Elementwise Operations

6.1 Elementary Mathematical Functions

Prototype	Description
<pre>void csipl_vacos_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vacos_f</code> <code>csipl_vacos_d</code></p>
<pre>void csipl_macos_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_macos_f</code> <code>csipl_macos_d</code></p>
<pre>void csipl_vasin_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vasin_f</code> <code>csipl_vasin_d</code></p>
<pre>void csipl_masin_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_masin_f</code> <code>csipl_masin_d</code></p>
<pre>void csipl_vatan_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vatan_f</code> <code>csipl_vatan_d</code></p>
<pre>void csipl_matan_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_matan_f</code> <code>csipl_matan_d</code></p>

Prototype	Description
<pre>void csipl_vatan2_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vatan2_f</code> <code>csipl_vatan2_d</code></p>
<pre>void csipl_matan2_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_matan2_f</code> <code>csipl_matan2_d</code></p>
<pre>void csipl_vcos_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the cosine for each element of a vector. Element angle values are in radians.</p> <p>The following instances are supported:</p> <p><code>csipl_vcos_f</code> <code>csipl_vcos_d</code></p>
<pre>void csipl_mcos_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the cosine for each element of a matrix. Element angle values are in radians.</p> <p>The following instances are supported:</p> <p><code>csipl_mcos_f</code> <code>csipl_mcos_d</code></p>
<pre>void csipl_vcosh_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the hyperbolic cosine for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vcosh_f</code> <code>csipl_vcosh_d</code></p>
<pre>void csipl_mcosh_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the hyperbolic cosine for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_mcosh_f</code> <code>csipl_mcosh_d</code></p>
<pre>void csipl_vexp_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the exponential function value for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vexp_f</code> <code>csipl_vexp_d</code></p>

Prototype	Description
<pre>void csipl_cvexp_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the exponential function value for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvexp_inter_f</code> <code>csipl_cvexp_inter_d</code></p>
<pre>void csipl_cvexp_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);</pre>	<p>Computes the exponential function value for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvexp_split_f</code> <code>csipl_cvexp_split_d</code></p>
<pre>void csipl_mexp_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the exponential function value for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mexp_f</code> <code>csipl_mexp_d</code></p>
<pre>void csipl_cmexp_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the exponential function value for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmexp_inter_f</code> <code>csipl_cmexp_inter_d</code></p>
<pre>void csipl_cmexp_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);</pre>	<p>Computes the exponential function value for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmexp_split_f</code> <code>csipl_cmexp_split_d</code></p>
<pre>void csipl_vexp10_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the base 10 exponential for each element of a vector. The following instances are supported:</p> <p><code>csipl_vexp10_f</code> <code>csipl_vexp10_d</code></p>
<pre>void csipl_mexp10_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the base 10 exponential for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mexp10_f</code> <code>csipl_mexp10_d</code></p>

Prototype	Description
<pre>void csipl_vfloor_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the floor for each element of a vector. The following instances are supported:</p> <p><code>csipl_vfloor_f</code> <code>csipl_vfloor_d</code></p>
<pre>void csipl_vlog_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the natural logarithm for each element of a vector. The following instances are supported:</p> <p><code>csipl_vlog_f</code> <code>csipl_vlog_d</code></p>
<pre>void csipl_cvlog_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the natural logarithm for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvlog_inter_f</code> <code>csipl_cvlog_inter_d</code></p>
<pre>void csipl_cvlog_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);</pre>	<p>Computes the natural logarithm for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvlog_split_f</code> <code>csipl_cvlog_split_d</code></p>
<pre>void csipl_mlog_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the natural logarithm for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mlog_f</code> <code>csipl_mlog_d</code></p>
<pre>void csipl_cmlog_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the natural logarithm for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmlog_inter_f</code> <code>csipl_cmlog_inter_d</code></p>
<pre>void csipl_cmlog_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);</pre>	<p>Computes the natural logarithm for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmlog_split_f</code> <code>csipl_cmlog_split_d</code></p>

Prototype	Description
<pre>void csipl_vlog10_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the base ten logarithm for each element of a vector. The following instances are supported:</p> <p><code>csipl_vlog10_f</code> <code>csipl_vlog10_d</code></p>
<pre>void csipl_mlog10_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the base ten logarithm for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mlog10_f</code> <code>csipl_mlog10_d</code></p>
<pre>void csipl_vsin_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the sine for each element of a vector. Element angle values are in radians. The following instances are supported:</p> <p><code>csipl_vsin_f</code> <code>csipl_vsin_d</code></p>
<pre>void csipl_msin_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the sine for each element of a matrix. Element angle values are in radians. The following instances are supported:</p> <p><code>csipl_msin_f</code> <code>csipl_msin_d</code></p>
<pre>void csipl_vsinh_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the hyperbolic sine for each element of a vector. The following instances are supported:</p> <p><code>csipl_vsinh_f</code> <code>csipl_vsinh_d</code></p>
<pre>void csipl_msinh_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the hyperbolic sine for each element of a matrix. The following instances are supported:</p> <p><code>csipl_msinh_f</code> <code>csipl_msinh_d</code></p>
<pre>void csipl_vsqrt_P(csipl_Dvview_P * A, csipl_stride strideA, csipl_Dvview_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the square root for each element of a vector. The following instances are supported:</p> <p><code>csipl_vsqrt_f</code> <code>csipl_vsqrt_d</code></p>
<pre>void csipl_cvsqrt_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the square root for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvsqrt_inter_f</code> <code>csipl_cvsqrt_inter_d</code></p>

Prototype	Description
<pre>void csipl_cvsqrt_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);</pre>	<p>Compute the square root for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cvsqrt_split_f</code> <code>csipl_cvsqrt_split_d</code></p>
<pre>void csipl_msqrt_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the square root for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_msqrt_f</code> <code>csipl_msqrt_d</code></p>
<pre>void csipl_cmsqrt_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the square root for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_cmsqrt_inter_f</code> <code>csipl_cmsqrt_inter_d</code></p>
<pre>void csipl_cmsqrt_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);</pre>	<p>Compute the square root for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_cmsqrt_split_f</code> <code>csipl_cmsqrt_split_d</code></p>
<pre>void csipl_mtanh_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the tangent for each element of a matrix. Element angle values are in radians.</p> <p>The following instances are supported:</p> <p><code>csipl_mtanh_f</code> <code>csipl_mtanh_d</code></p>
<pre>void csipl_mtanh_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the hyperbolic tangent for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_mtanh_f</code> <code>csipl_mtanh_d</code></p>

6.2 Unary Operations

Prototype	Description
<pre>void csipl_varg_P(void * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the argument in radians $[-\pi, \pi]$ for each element of a complex vector.</p> <p>The following instances are supported:</p> <p><code>csipl_varg_f</code> <code>csipl_varg_d</code></p>
<pre>void csipl_marg_P(void * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the argument in radians $[-\pi, \pi]$ for each element of a complex matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_marg_f</code> <code>csipl_marg_d</code></p>
<pre>void csipl_vceil_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the ceiling for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vceil_f</code> <code>csipl_vceil_d</code></p>
<pre>void csipl_cvconj_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the conjugate for each element of a complex vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cvconj_inter_f</code> <code>csipl_cvconj_inter_d</code></p>
<pre>void csipl_cvconj_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);</pre>	<p>Compute the conjugate for each element of a complex vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cvconj_split_f</code> <code>csipl_cvconj_split_d</code></p>
<pre>void csipl_cmconj_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the conjugate for each element of a complex matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_cmconj_inter_f</code> <code>csipl_cmconj_inter_d</code></p>
<pre>void csipl_cmconj_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);</pre>	<p>Compute the conjugate for each element of a complex matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_cmconj_split_f</code> <code>csipl_cmconj_split_d</code></p>

Prototype	Description
<pre>void csipl_vcumsum_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the cumulative sum of the elements of a vector. The following instances are supported:</p> <p>csipl_vcumsum_f csipl_vcumsum_d csipl_vcumsum_i</p>
<pre>void csipl_cvcumsum_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the cumulative sum of the elements of a vector. The following instances are supported:</p> <p>csipl_cvcumsum_inter_f csipl_cvcumsum_inter_d csipl_cvcumsum_inter_i</p>
<pre>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);</pre>	<p>Compute the cumulative sum of the elements of a vector. The following instances are supported:</p> <p>csipl_cvcumsum_split_f csipl_cvcumsum_split_d csipl_cvcumsum_split_i</p>
<pre>void csipl_mcumsum_P(csipl_major dir, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the cumulative sums of the elements in the rows or columns of a matrix. The following instances are supported:</p> <p>csipl_mcumsum_f csipl_mcumsum_d csipl_mcumsum_i</p>
<pre>void csipl_cmcumsum_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the cumulative sum of the elements of a vector. The following instances are supported:</p> <p>csipl_cmcumsum_inter_f csipl_cmcumsum_inter_d csipl_cmcumsum_inter_i</p>
<pre>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);</pre>	<p>Compute the cumulative sum of the elements of a vector. The following instances are supported:</p> <p>csipl_cmcumsum_split_f csipl_cmcumsum_split_d csipl_cmcumsum_split_i</p>
<pre>void csipl_veuler_P(scalar_P * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a vector. The following instances are supported:</p> <p>csipl_veuler_f csipl_veuler_d</p>

Prototype	Description
<pre>void csipl_meuler_P(scalar_P * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_meuler_f</code> <code>csipl_meuler_d</code></p>
<pre>void csipl_vmag_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the magnitude for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vmag_f</code> <code>csipl_vmag_d</code> <code>csipl_vmag_i</code> <code>csipl_vmag_si</code></p>
<pre>void csipl_cvmag_inter_P(void * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the magnitude for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cvmag_inter_f</code> <code>csipl_cvmag_inter_d</code></p>
<pre>void csipl_cvmag_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the magnitude for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cvmag_split_f</code> <code>csipl_cvmag_split_d</code></p>
<pre>void csipl_mmag_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the magnitude for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_mmag_f</code> <code>csipl_mmag_d</code></p>
<pre>void csipl_cmmag_P(void * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the magnitude for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_cmmag_inter_f</code> <code>csipl_cmmag_inter_d</code></p>
<pre>void csipl_cmmag_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the magnitude for each element of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_cmmag_split_f</code> <code>csipl_cmmag_split_d</code></p>

Prototype	Description
<pre>void csipl_vcmagsq_inter_P(void * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the square of the magnitudes for each element of a vector. The following instances are supported:</p> <p><code>csipl_vcmagsq_inter_f</code> <code>csipl_vcmagsq_inter_d</code></p>
<pre>void csipl_vcmagsq_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the square of the magnitudes for each element of a vector. The following instances are supported:</p> <p><code>csipl_vcmagsq_split_f</code> <code>csipl_vcmagsq_split_d</code></p>
<pre>void csipl_mcmagsq_P(void * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the square of the magnitudes for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mcmagsq_f</code> <code>csipl_mcmagsq_d</code></p>
<pre>scalar_P csipl_vmeanval_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the mean value of the elements of a vector. The following instances are supported:</p> <p><code>csipl_vmeanval_f</code> <code>csipl_vmeanval_d</code></p>
<pre>csipl_cscalar_P csipl_cvmeanval_inter_P(void * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the mean value of the elements of a vector. The following instances are supported:</p> <p><code>csipl_cvmeanval_inter_f</code> <code>csipl_cvmeanval_inter_d</code></p>
<pre>csipl_cscalar_P csipl_cvmeanval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the mean value of the elements of a vector. The following instances are supported:</p> <p><code>csipl_cvmeanval_split_f</code> <code>csipl_cvmeanval_split_d</code></p>
<pre>scalar_P csipl_mmeanval_P(scalar_P * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the mean value of the elements of a matrix. The following instances are supported:</p> <p><code>csipl_mmeanval_f</code> <code>csipl_mmeanval_d</code></p>
<pre>csipl_cscalar_P csipl_cmmeanval_inter_P(void * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the mean value of the elements of a matrix. The following instances are supported:</p> <p><code>csipl_cmmeanval_inter_f</code> <code>csipl_cmmeanval_inter_d</code></p>
<pre>csipl_cscalar_P csipl_cmmeanval_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the mean value of the elements of a matrix. The following instances are supported:</p> <p><code>csipl_cmmeanval_split_f</code> <code>csipl_cmmeanval_split_d</code></p>

Prototype	Description
<pre>scalar_P csipl_vmeansqval_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the mean magnitude squared value of the elements of a vector. The following instances are supported:</p> <p><code>csipl_vmeansqval_f</code> <code>csipl_vmeansqval_d</code></p>
<pre>scalar_P csipl_cvmeansqval_inter_P(void * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the mean magnitude squared value of the elements of a vector. The following instances are supported:</p> <p><code>csipl_cvmeansqval_inter_f</code> <code>csipl_cvmeansqval_inter_d</code></p>
<pre>scalar_P csipl_cvmeansqval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the mean magnitude squared value of the elements of a vector. The following instances are supported:</p> <p><code>csipl_cvmeansqval_split_f</code> <code>csipl_cvmeansqval_split_d</code></p>
<pre>scalar_P csipl_mmeansqval_P(scalar_P * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the mean magnitude squared value of the elements of a matrix. The following instances are supported:</p> <p><code>csipl_mmeansqval_f</code> <code>csipl_mmeansqval_d</code></p>
<pre>scalar_P csipl_cmmeansqval_inter_P(void * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the mean magnitude squared value of the elements of a matrix. The following instances are supported:</p> <p><code>csipl_cmmeansqval_inter_f</code> <code>csipl_cmmeansqval_inter_d</code></p>
<pre>scalar_P csipl_cmmeansqval_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the mean magnitude squared value of the elements of a matrix. The following instances are supported:</p> <p><code>csipl_cmmeansqval_split_f</code> <code>csipl_cmmeansqval_split_d</code></p>
<pre>scalar_P csipl_vmodulate_P(scalar_P * A, csipl_stride strideA, scalar_P nu, scalar_P phi, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the modulation of a real vector by a specified complex frequency. The following instances are supported:</p> <p><code>csipl_vmodulate_f</code> <code>csipl_vmodulate_d</code></p>
<pre>scalar_P csipl_cvmodulate_inter_P(void * A, csipl_stride strideA, scalar_P nu, scalar_P phi, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the modulation of a real vector by a specified complex frequency. The following instances are supported:</p> <p><code>csipl_cvmodulate_inter_f</code> <code>csipl_cvmodulate_inter_d</code></p>

Prototype	Description
<pre> scalar_P csipl_cvmodulate_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P nu, scalar_P phi, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n); </pre>	<p>Computes the modulation of a real vector by a specified complex frequency. The following instances are supported:</p> <p><code>csipl_cvmodulate_split_f</code> <code>csipl_cvmodulate_split_d</code></p>
<pre> void csipl_vneg_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n); </pre>	<p>Computes the negation for each element of a vector. The following instances are supported:</p> <p><code>csipl_vneg_f</code> <code>csipl_vneg_d</code> <code>csipl_vneg_i</code> <code>csipl_vneg_si</code></p>
<pre> void csipl_cvneg_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n); </pre>	<p>Computes the negation for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvneg_inter_f</code> <code>csipl_cvneg_inter_d</code></p>
<pre> void csipl_cvneg_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); </pre>	<p>Computes the negation for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvneg_split_f</code> <code>csipl_cvneg_split_d</code></p>
<pre> void csipl_mneg_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n); </pre>	<p>Computes the negation for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mneg_f</code> <code>csipl_mneg_d</code> <code>csipl_mneg_i</code></p>
<pre> void csipl_cmneg_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n); </pre>	<p>Computes the negation for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmneg_inter_f</code> <code>csipl_cmneg_inter_d</code></p>
<pre> void csipl_cmneg_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); </pre>	<p>Computes the negation for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmneg_split_f</code> <code>csipl_cmneg_split_d</code></p>

Prototype	Description
<pre>void csipl_vrecip_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a vector. The following instances are supported:</p> <p><code>csipl_vrecip_f</code> <code>csipl_vrecip_d</code></p>
<pre>void csipl_cvrecip_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvrecip_inter_f</code> <code>csipl_cvrecip_inter_d</code></p>
<pre>void csipl_cvrecip_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);</pre>	<p>Computes the reciprocal for each element of a vector. The following instances are supported:</p> <p><code>csipl_cvrecip_split_f</code> <code>csipl_cvrecip_split_d</code></p>
<pre>void csipl_mrecip_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mrecip_f</code> <code>csipl_mrecip_d</code></p>
<pre>void csipl_cmrecip_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmrecip_inter_f</code> <code>csipl_cmrecip_inter_d</code></p>
<pre>void csipl_cmrecip_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);</pre>	<p>Computes the reciprocal for each element of a matrix. The following instances are supported:</p> <p><code>csipl_cmrecip_split_f</code> <code>csipl_cmrecip_split_d</code></p>
<pre>void csipl_vrsqrt_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the reciprocal of the square root for each element of a vector. The following instances are supported:</p> <p><code>csipl_vrsqrt_f</code> <code>csipl_vrsqrt_d</code></p>

Prototype	Description
<pre>void csipl_mrsqrt_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the reciprocal of the square root for each element of a matrix. The following instances are supported:</p> <p><code>csipl_mrsqrt_f</code> <code>csipl_mrsqrt_d</code></p>
<pre>void csipl_vsqr_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the square for each element of a vector. The following instances are supported:</p> <p><code>csipl_vsqr_f</code> <code>csipl_vsqr_d</code></p>
<pre>void csipl_msqr_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the square for each element of a matrix. The following instances are supported:</p> <p><code>csipl_msqr_f</code> <code>csipl_msqr_d</code></p>
<pre>scalar_P csipl_cvsumval_inter_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector. The following instances are supported:</p> <p><code>csipl_cvsumval_inter_f</code> <code>csipl_cvsumval_inter_d</code> <code>csipl_cvsumval_inter_i</code></p>
<pre>scalar_P csipl_cvsumval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector. The following instances are supported:</p> <p><code>csipl_cvsumval_split_f</code> <code>csipl_cvsumval_split_d</code> <code>csipl_cvsumval_split_i</code></p>
<pre>scalar_P csipl_msumval_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector. The following instances are supported:</p> <p><code>csipl_msumval_f</code> <code>csipl_msumval_d</code> <code>csipl_msumval_i</code></p>
<pre>scalar_P csipl_csumval_inter_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector. The following instances are supported:</p> <p><code>csipl_csumval_inter_f</code> <code>csipl_csumval_inter_d</code> <code>csipl_csumval_inter_i</code></p>
<pre>scalar_P csipl_csumval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector. The following instances are supported:</p> <p><code>csipl_csumval_split_f</code> <code>csipl_csumval_split_d</code> <code>csipl_csumval_split_i</code></p>

Prototype	Description
<pre>scalar_P csipl_msumsqval_P(scalar_P * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the sum of the squares of the elements of a matrix. The following instances are supported:</p> <pre>csipl_msumsqval_f csipl_msumsqval_d</pre>

6.3 Binary Operations

Prototype	Description
<pre>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);</pre>	<p>Computes the sum, by element, of two vectors. The following instances are supported:</p> <pre>csipl_vadd_f csipl_vadd_d csipl_vadd_i csipl_vadd_si</pre>
<pre>void csipl_cvadd_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of two vectors. The following instances are supported:</p> <pre>csipl_cvadd_inter_f csipl_cvadd_inter_d csipl_cvadd_inter_i</pre>
<pre>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);</pre>	<p>Computes the sum, by element, of two vectors. The following instances are supported:</p> <pre>csipl_cvadd_split_f csipl_cvadd_split_d csipl_cvadd_split_i</pre>
<pre>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);</pre>	<p>Computes the sum, by element, of two matrices. The following instances are supported:</p> <pre>csipl_madd_f csipl_madd_d csipl_madd_i</pre>
<pre>void csipl_cmadd_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of two matrices. The following instances are supported:</p> <pre>csipl_cmadd_inter_f csipl_cmadd_inter_d</pre>

Prototype	Description
<pre>void csipl_cmadd_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);</pre>	<p>Computes the sum, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_cmadd_split_f csipl_cmadd_split_d</pre>
<pre>void csipl_rcvadd_P(scalar_P * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_rcvadd_inter_f csipl_rcvadd_inter_d</pre>
<pre>void csipl_rcvadd_split_P(scalar_P * A, 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);</pre>	<p>Computes the sum, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_rcvadd_split_f csipl_rcvadd_split_d</pre>
<pre>void csipl_rcmadd_inter_P(scalar_P * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_rcmadd_inter_f csipl_rcmadd_inter_d</pre>
<pre>void csipl_rcmadd_split_P(scalar_P * A, 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);</pre>	<p>Computes the sum, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_rcmadd_split_f csipl_rcmadd_split_d</pre>

Prototype	Description
<pre>void csipl_svadd_P(scalar_P a, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a vector. The following instances are supported:</p> <pre>csipl_svadd_f csipl_svadd_d csipl_svadd_i csipl_svadd_si</pre>
<pre>void csipl_csvadd_inter_P(csipl_cscalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a vector. The following instances are supported:</p> <pre>csipl_csvadd_inter_f csipl_csvadd_inter_d</pre>
<pre>void csipl_csvadd_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, 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);</pre>	<p>Computes the sum, by element, of a scalar and a vector. The following instances are supported:</p> <pre>csipl_csvadd_split_f csipl_csvadd_split_d</pre>
<pre>void csipl_smadd_P(scalar_P a, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a matrix. The following instances are supported:</p> <pre>csipl_smadd_f csipl_smadd_d csipl_smadd_i</pre>
<pre>void csipl_csmadd_inter_P(csipl_cscalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a matrix. The following instances are supported:</p> <pre>csipl_csmadd_inter_f csipl_csmadd_inter_d</pre>
<pre>void csipl_csmadd_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, 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);</pre>	<p>Computes the sum, by element, of a scalar and a matrix. The following instances are supported:</p> <pre>csipl_csmadd_split_f csipl_csmadd_split_d</pre>

Prototype	Description
<pre>void csipl_rscvadd_inter_P(scalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvadd_inter_f</code> <code>csipl_rscvadd_inter_d</code></p>
<pre>void csipl_rscvadd_split_P(scalar_P a, 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);</pre>	<p>Computes the sum, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvadd_split_f</code> <code>csipl_rscvadd_split_d</code></p>
<pre>void csipl_rscmadd_inter_P(scalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <p><code>csipl_rscmadd_inter_f</code> <code>csipl_rscmadd_inter_d</code></p>
<pre>void csipl_rscmadd_split_P(scalar_P a, 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);</pre>	<p>Computes the sum, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <p><code>csipl_rscmadd_split_f</code> <code>csipl_rscmadd_split_d</code></p>
<pre>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);</pre>	<p>Computes the quotient, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_vdiv_f</code> <code>csipl_vdiv_d</code> <code>csipl_svdiv_f</code> <code>csipl_svdiv_d</code></p>
<pre>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);</pre>	<p>Computes the quotient, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_crvdiv_inter_f</code> <code>csipl_crvdiv_inter_d</code> <code>csipl_cvdiv_inter_f</code> <code>csipl_cvdiv_inter_d</code></p>

Prototype	Description
<pre>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);</pre>	<p>Computes the quotient, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_rcvdiv_split_f csipl_rcvdiv_split_d csipl_cvdiv_split_f csipl_cvdiv_split_d</pre>
<pre>void csipl_mdiv_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mdiv_f csipl_mdiv_d</pre>
<pre>void csipl_cmdiv_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_cmdiv_inter_f csipl_cmdiv_inter_d</pre>
<pre>void csipl_cmdiv_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);</pre>	<p>Computes the quotient, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_cmdiv_split_f csipl_cmdiv_split_d</pre>
<pre>void csipl_rcvdiv_inter_P(scalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_rcvdiv_inter_f csipl_rcvdiv_inter_d</pre>
<pre>void csipl_rcvdiv_split_P(scalar_P a, 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);</pre>	<p>Computes the quotient, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_rcvdiv_split_f csipl_rcvdiv_split_d</pre>

Prototype	Description
<pre>void csipl_rcmdiv_inter_P(scalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_rcmdiv_inter_f</code> <code>csipl_rcmdiv_inter_d</code></p>
<pre>void csipl_rcmdiv_split_P(scalar_P a, 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);</pre>	<p>Computes the quotient, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_rcmdiv_split_f</code> <code>csipl_rcmdiv_split_d</code></p>
<pre>void csipl_crmdiv_inter_P(void * A, int ldA, scalar_P * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_crmdiv_inter_f</code> <code>csipl_crmdiv_inter_d</code></p>
<pre>void csipl_crmdiv_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B, int ldB, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_crmdiv_split_f</code> <code>csipl_crmdiv_split_d</code></p>
<pre>void csipl_rscmsub_inter_P(scalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <p><code>csipl_rscmsub_inter_f</code> <code>csipl_rscmsub_inter_d</code></p>
<pre>void csipl_rscmsub_split_P(scalar_P a, 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);</pre>	<p>Computes the difference, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <p><code>csipl_rscmsub_split_f</code> <code>csipl_rscmsub_split_d</code></p>

Prototype	Description
<pre>void csipl_vsdiv_P(scalar_P * A, csipl_stride strideA, scalar_P b, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of a vector and a scalar. The following instances are supported:</p> <p><code>csipl_vsdiv_f</code> <code>csipl_vsdiv_d</code></p>
<pre>void csipl_cvrdiv_inter_P(scalar_P * A, csipl_stride strideA, scalar_P b, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of a vector and a scalar. The following instances are supported:</p> <p><code>csipl_cvrdiv_inter_f</code> <code>csipl_cvrdiv_inter_d</code></p>
<pre>void csipl_cvrdiv_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P b_re, scalar_P b_im, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of a vector and a scalar. The following instances are supported:</p> <p><code>csipl_cvrdiv_split_f</code> <code>csipl_cvrdiv_split_d</code></p>
<pre>void csipl_msdiv_P(scalar_P * A, int ldA, scalar_P b, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of a matrix and a scalar. The following instances are supported:</p> <p><code>csipl_msdiv_f</code> <code>csipl_msdiv_d</code></p>
<pre>void csipl_cmrsdiv_inter_P(void * A, int ldA, csipl_cscalar_P b, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of a matrix and a scalar. The following instances are supported:</p> <p><code>csipl_cmrsdiv_inter_f</code> <code>csipl_cmrsdiv_inter_d</code></p>
<pre>void csipl_cmrsdiv_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, csipl_cscalar_P b_re, csipl_cscalar_P b_im, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of a matrix and a scalar. The following instances are supported:</p> <p><code>csipl_cmrsdiv_split_f</code> <code>csipl_cmrsdiv_split_d</code></p>

Prototype	Description
<pre>void csipl_vexpoavg_P(scalar_P a, scalar_P * B, csipl_stride strideB, scalar_P * C, csipl_stride strideC, csipl_length n);</pre>	<p>Computes an exponential weighted average, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_vexpoavg_f</code> <code>csipl_vexpoavg_d</code></p>
<pre>void csipl_cvexpoavg_inter_P(scalar_P a, void * B, csipl_stride strideB, void * C, csipl_stride strideC, csipl_length n);</pre>	<p>Computes an exponential weighted average, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_cvexpoavg_inter_f</code> <code>csipl_cvexpoavg_inter_d</code></p>
<pre>void csipl_cvexpoavg_split_P(scalar_P a, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, scalar_P * C_re, scalar_P * C_im, csipl_stride strideC, csipl_length n);</pre>	<p>Computes an exponential weighted average, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_cvexpoavg_split_f</code> <code>csipl_cvexpoavg_split_d</code></p>
<pre>void csipl_mexpoavg_P(scalar_P a, scalar_P * B, int ldB, scalar_P * C, int ldC, csipl_length m, csipl_length n);</pre>	<p>Computes an exponential weighted average, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_mexpoavg_f</code> <code>csipl_mexpoavg_d</code></p>
<pre>void csipl_cmexpoavg_inter_P(scalar_P a, void * B, int ldB, void * C, int ldC, csipl_length m, csipl_length n);</pre>	<p>Computes an exponential weighted average, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_cmexpoavg_inter_f</code> <code>csipl_cmexpoavg_inter_d</code></p>
<pre>void csipl_cmexpoavg_split_P(scalar_P a, scalar_P * B_re, scalar_P * B_im, int ldB, scalar_P * C_re, scalar_P * C_im, int ldC, csipl_length m, csipl_length n);</pre>	<p>Computes an exponential weighted average, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_cmexpoavg_split_f</code> <code>csipl_cmexpoavg_split_d</code></p>

Prototype	Description
<pre>void csipl_vhypot_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the square root of the sum of squares, by element, of two input vectors.</p> <p>The following instances are supported:</p> <p>csipl_vhypot_f csipl_vhypot_d</p>
<pre>void csipl_mhypot_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the square root of the sum of squares, by element, of two input matrices.</p> <p>The following instances are supported:</p> <p>csipl_mhypot_f csipl_mhypot_d</p>
<pre>void csipl_cvjmul_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of a complex vector with the conjugate of a second complex vector, by element.</p> <p>The following instances are supported:</p> <p>csipl_cvjmul_inter_f csipl_cvjmul_inter_d</p>
<pre>void csipl_cvjmul_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);</pre>	<p>Computes the product of a complex vector with the conjugate of a second complex vector, by element.</p> <p>The following instances are supported:</p> <p>csipl_cvjmul_split_f csipl_cvjmul_split_d</p>
<pre>void csipl_cmjmul_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.</p> <p>The following instances are supported:</p> <p>csipl_cmjmul_inter_f csipl_cmjmul_inter_d</p>
<pre>void csipl_cmjmul_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);</pre>	<p>Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.</p> <p>The following instances are supported:</p> <p>csipl_cmjmul_split_f csipl_cmjmul_split_d</p>

Prototype	Description
<pre>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);</pre>	<p>Computes the product, by element, of two vectors. The following instances are supported:</p> <pre>csipl_vmul_f csipl_vmul_d csipl_vmul_i csipl_vmul_si csipl_svmul_f csipl_svmul_d csipl_svmul_i csipl_svmul_si</pre>
<pre>void csipl_cvmul_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of two vectors. The following instances are supported:</p> <pre>csipl_cvmul_inter_f csipl_cvmul_inter_d</pre>
<pre>void csipl_cvmul_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);</pre>	<p>Computes the product, by element, of two vectors. The following instances are supported:</p> <pre>csipl_cvmul_split_f csipl_cvmul_split_d</pre>
<pre>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);</pre>	<p>Computes the product, by element, of two matrices. The following instances are supported:</p> <pre>csipl_mmul_f csipl_mmul_d csipl_mmul_i</pre>
<pre>void csipl_cmmul_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of two matrices. The following instances are supported:</p> <pre>csipl_cmmul_inter_f csipl_cmmul_inter_d</pre>

Prototype	Description
<pre>void csipl_cmmul_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);</pre>	<p>Computes the product, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_cmmul_split_f</code> <code>csipl_cmmul_split_d</code></p>
<pre>void csipl_rcvmul_inter_P(scalar_P * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_rcvmul_inter_f</code> <code>csipl_rcvmul_inter_d</code></p>
<pre>void csipl_rcvmul_split_P(scalar_P * A, 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);</pre>	<p>Computes the product, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_rcvmul_split_f</code> <code>csipl_rcvmul_split_d</code></p>
<pre>void csipl_rcmmul_inter_P(scalar_P * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_rcmmul_inter_f</code> <code>csipl_rcmmul_inter_d</code></p>
<pre>void csipl_rcmmul_split_P(scalar_P * A, 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);</pre>	<p>Computes the product, by element, of two matrices. The following instances are supported:</p> <p><code>csipl_rcmmul_split_f</code> <code>csipl_rcmmul_split_d</code></p>
<pre>void csipl_rscvmul_inter_P(scalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvmul_inter_f</code> <code>csipl_rscvmul_inter_d</code></p>

Prototype	Description
<pre>void csipl_rscvmul_split_P(scalar_P a, 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);</pre>	<p>Computes the product, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvmul_split_f</code> <code>csipl_rscvmul_split_d</code></p>
<pre>void csipl_csvmul_inter_P(csipl_cscalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of a scalar and a vector. The following instances are supported:</p> <p><code>csipl_csvmul_inter_f</code> <code>csipl_csvmul_inter_d</code></p>
<pre>void csipl_csvmul_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, 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);</pre>	<p>Computes the product, by element, of a scalar and a vector. The following instances are supported:</p> <p><code>csipl_csvmul_split_f</code> <code>csipl_csvmul_split_d</code></p>
<pre>void csipl_smmul_P(scalar_P a, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_smmul_f</code> <code>csipl_smmul_d</code></p>
<pre>void csipl_csmmul_inter_P(csipl_cscalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_csmmul_inter_f</code> <code>csipl_csmmul_inter_d</code></p>
<pre>void csipl_csmmul_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, 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);</pre>	<p>Computes the product, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_csmmul_split_f</code> <code>csipl_csmmul_split_d</code></p>

Prototype	Description
<pre>void csipl_rscmmul_inter_P(scalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <pre>csipl_rscmmul_inter_f csipl_rscmmul_inter_d</pre>
<pre>void csipl_rscmmul_split_P(scalar_P a, 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);</pre>	<p>Computes the product, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <pre>csipl_rscmmul_split_f csipl_rscmmul_split_d</pre>
<pre>void csipl_vmmul_P(scalar_P * A, csipl_stride strideA, scalar_P * B, int ldB, csipl_major major, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported:</p> <pre>csipl_vmmul_f csipl_vmmul_d</pre>
<pre>void csipl_cvmmul_inter_P(void * A, csipl_stride strideA, void * B, int ldB, csipl_major major, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported:</p> <pre>csipl_cvmmul_inter_f csipl_cvmmul_inter_d</pre>
<pre>void csipl_cvmmul_split_P(csipl_Dvview_P * A_re, csipl_Dvview_P * A_im, csipl_stride strideA, csipl_Dmview_P * B_re, csipl_Dmview_P * B_im, int ldB, csipl_major major, csipl_Dmview_P * R_re, csipl_Dmview_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported:</p> <pre>csipl_cvmmul_split_f csipl_cvmmul_split_d</pre>

Prototype	Description
<pre>void csipl_rvcmmul_inter_P(scalar_P * A, csipl_stride strideA, void * B, int ldB, csipl_major major, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_rvcmmul_inter_f</code> <code>csipl_rvcmmul_inter_d</code></p>
<pre>void csipl_rvcmmul_split_P(scalar_P * A, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, int ldB, csipl_major major, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_rvcmmul_split_f</code> <code>csipl_rvcmmul_split_d</code></p>
<pre>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);</pre>	<p>Computes the difference, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vsub_f</code> <code>csipl_vsub_d</code> <code>csipl_vsub_i</code> <code>csipl_vsub_si</code></p>
<pre>void csipl_cvsub_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_cvsub_inter_f</code> <code>csipl_cvsub_inter_d</code></p>
<pre>void csipl_cvsub_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);</pre>	<p>Computes the difference, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_cvsub_split_f</code> <code>csipl_cvsub_split_d</code></p>

Prototype	Description
<pre>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);</pre>	<p>Computes the difference, by element, of two matrices. The following instances are supported:</p> <pre>csipl_msub_f csipl_msub_d csipl_msub_i</pre>
<pre>void csipl_cmsub_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices. The following instances are supported:</p> <pre>csipl_cmsub_inter_f csipl_cmsub_inter_d</pre>
<pre>void csipl_cmsub_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);</pre>	<p>Computes the difference, by element, of two matrices. The following instances are supported:</p> <pre>csipl_cmsub_split_f csipl_cmsub_split_d</pre>
<pre>void csipl_crmsub_inter_P(void * A, int ldA, scalar_P * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices. The following instances are supported:</p> <pre>csipl_crmsub_inter_f csipl_crmsub_inter_d</pre>
<pre>void csipl_crmsub_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B, int ldB, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices. The following instances are supported:</p> <pre>csipl_crmsub_split_f csipl_crmsub_split_d</pre>

Prototype	Description
<pre>void csipl_rcvsub_inter_P(scalar_P * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_rcvsub_inter_f</code> <code>csipl_rcvsub_inter_d</code></p>
<pre>void csipl_rcvsub_split_P(scalar_P * A, 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);</pre>	<p>Computes the difference, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_rcvsub_split_f</code> <code>csipl_rcvsub_split_d</code></p>
<pre>void csipl_rcmsub_inter_P(scalar_P * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_rcmsub_inter_f</code> <code>csipl_rcmsub_inter_d</code></p>
<pre>void csipl_rcmsub_split_P(scalar_P * A, 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);</pre>	<p>Computes the difference, by element, of two matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_rcmsub_split_f</code> <code>csipl_rcmsub_split_d</code></p>
<pre>void csipl_crvsub_inter_P(void * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_crvsub_inter_f</code> <code>csipl_crvsub_inter_d</code></p>
<pre>void csipl_crvsub_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_crvsub_split_f</code> <code>csipl_crvsub_split_d</code></p>

Prototype	Description
<pre>void csipl_svsub_P(scalar_P a, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a vector. The following instances are supported:</p> <p><code>csipl_svsub_f</code> <code>csipl_svsub_d</code> <code>csipl_svsub_i</code> <code>csipl_svsub_si</code></p>
<pre>void csipl_csvsub_inter_P(csipl_cscalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a vector. The following instances are supported:</p> <p><code>csipl_csvsub_inter_f</code> <code>csipl_csvsub_inter_d</code></p>
<pre>void csipl_csvsub_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, 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);</pre>	<p>Computes the difference, by element, of a scalar and a vector. The following instances are supported:</p> <p><code>csipl_csvsub_split_f</code> <code>csipl_csvsub_split_d</code></p>
<pre>void csipl_smsub_P(scalar_P a, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_smsub_f</code> <code>csipl_smsub_d</code> <code>csipl_smsub_i</code></p>
<pre>void csipl_csmsub_inter_P(csipl_cscalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_csmsub_inter_f</code> <code>csipl_csmsub_inter_d</code></p>
<pre>void csipl_csmsub_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, 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);</pre>	<p>Computes the difference, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_csmsub_split_f</code> <code>csipl_csmsub_split_d</code></p>

Prototype	Description
<pre>void csipl_smdiv_P(scalar_P a, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_smdiv_f</code> <code>csipl_smdiv_d</code></p>
<pre>void csipl_csmdiv_inter_P(csipl_cscalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_csmdiv_inter_f</code> <code>csipl_csmdiv_inter_d</code></p>
<pre>void csipl_csmdiv_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, 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);</pre>	<p>Computes the quotient, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_csmdiv_split_f</code> <code>csipl_csmdiv_split_d</code></p>
<pre>void csipl_rscvdiv_inter_P(scalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvdiv_inter_f</code> <code>csipl_rscvdiv_inter_d</code></p>
<pre>void csipl_rscvdiv_split_P(scalar_P a, 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);</pre>	<p>Computes the quotient, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvdiv_split_f</code> <code>csipl_rscvdiv_split_d</code></p>
<pre>void csipl_rscvsub_inter_P(scalar_P a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvsub_inter_f</code> <code>csipl_rscvsub_inter_d</code></p>

Prototype	Description
<pre>void csipl_rscvsub_split_P(scalar_P a, 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);</pre>	<p>Computes the difference, by element, of a real scalar and a complex vector. The following instances are supported:</p> <p><code>csipl_rscvsub_split_f</code> <code>csipl_rscvsub_split_d</code></p>
<pre>void csipl_rscmdiv_inter_P(scalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <p><code>csipl_rscmdiv_inter_f</code> <code>csipl_rscmdiv_inter_d</code></p>
<pre>void csipl_rscmdiv_split_P(scalar_P a, 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);</pre>	<p>Computes the quotient, by element, of a real scalar and a complex matrix. The following instances are supported:</p> <p><code>csipl_rscmdiv_split_f</code> <code>csipl_rscmdiv_split_d</code></p>

6.4 Ternary Operations

Prototype	Description
<pre>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);</pre>	<p>Computes the sum of two vectors and product of a third vector, by element. The following instances are supported:</p> <p><code>csipl_vam_f</code> <code>csipl_vam_d</code> <code>csipl_vma_f</code> <code>csipl_vma_d</code></p>
<pre>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);</pre>	<p>Computes the sum of two vectors and product of a third vector, by element. The following instances are supported:</p> <p><code>csipl_cvam_inter_f</code> <code>csipl_cvam_inter_d</code> <code>csipl_cvma_inter_f</code> <code>csipl_cvma_inter_d</code></p>

Prototype	Description
<pre>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);</pre>	<p>Computes the sum of two vectors and product of a third vector, by element. The following instances are supported:</p> <pre>csipl_cvam_split_f csipl_cvam_split_d csipl_cvma_split_f csipl_cvma_split_d</pre>
<pre>void csipl_vmsa_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P c, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and sum of a scalar, by element. The following instances are supported:</p> <pre>csipl_vmsa_f csipl_vmsa_d</pre>
<pre>void csipl_cvmsa_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_cscalar_P c, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and sum of a scalar, by element. The following instances are supported:</p> <pre>csipl_cvmsa_inter_f csipl_cvmsa_inter_d</pre>
<pre>void csipl_cvmsa_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, csipl_cscalar_P c_re, csipl_cscalar_P c_im, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and sum of a scalar, by element. The following instances are supported:</p> <pre>csipl_cvmsa_split_f csipl_cvmsa_split_d</pre>
<pre>void csipl_vmsb_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);</pre>	<p>Computes the product of two vectors and difference of a third vector, by element. The following instances are supported:</p> <pre>csipl_vmsb_f csipl_vmsb_d</pre>

Prototype	Description
<pre>void csipl_cvmsb_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);</pre>	<p>Computes the product of two vectors and difference of a third vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_cvmsb_inter_f</code> <code>csipl_cvmsb_inter_d</code></p>
<pre>void csipl_cvmsb_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);</pre>	<p>Computes the product of two vectors and difference of a third vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_cvmsb_split_f</code> <code>csipl_cvmsb_split_d</code></p>
<pre>void csipl_vsam_P(scalar_P * A, csipl_stride strideA, scalar_P b, scalar_P * C, csipl_stride strideC, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum of a vector and a scalar, and product with a second vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_vsam_f</code> <code>csipl_vsam_d</code></p>
<pre>void csipl_cvсам_inter_P(void * A, csipl_stride strideA, csipl_cscalar_P b, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum of a vector and a scalar, and product with a second vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_cvсам_inter_f</code> <code>csipl_cvсам_inter_d</code></p>
<pre>void csipl_cvсам_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_cscalar_P b_re, csipl_cscalar_P b_im, 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);</pre>	<p>Computes the sum of a vector and a scalar, and product with a second vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_cvсам_split_f</code> <code>csipl_cvсам_split_d</code></p>

Prototype	Description
<pre>void csipl_vsbm_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);</pre>	<p>Computes the difference of two vectors, and product with a third vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_vsbm_f</code> <code>csipl_vsbm_d</code></p>
<pre>void csipl_cvsvbm_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);</pre>	<p>Computes the difference of two vectors, and product with a third vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_cvsvbm_inter_f</code> <code>csipl_cvsvbm_inter_d</code></p>
<pre>void csipl_cvsvbm_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);</pre>	<p>Computes the difference of two vectors, and product with a third vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_cvsvbm_split_f</code> <code>csipl_cvsvbm_split_d</code></p>
<pre>void csipl_vsma_P(scalar_P * A, csipl_stride strideA, scalar_P b, scalar_P * C, csipl_stride strideC, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_vsma_f</code> <code>csipl_vsma_d</code></p>
<pre>void csipl_cvsvma_inter_P(void * A, csipl_stride strideA, csipl_cscalar_P b, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second vector, by element.</p> <p>The following instances are supported:</p> <p><code>csipl_cvsvma_inter_f</code> <code>csipl_cvsvma_inter_d</code></p>

Prototype	Description
<pre>void csipl_cvmsa_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_cscalar_P b_re, csipl_cscalar_P b_im, 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);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second vector, by element.</p> <p>The following instances are supported:</p> <pre>csipl_cvmsa_split_f csipl_cvmsa_split_d</pre>
<pre>void csipl_vsmsa_P(scalar_P * A, csipl_stride strideA, scalar_P b, scalar_P c, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second scalar, by element.</p> <p>The following instances are supported:</p> <pre>csipl_vsmsa_f csipl_vsmsa_d</pre>
<pre>void csipl_cvmsa_inter_P(void * A, csipl_stride strideA, csipl_cscalar_P b, csipl_cscalar_P c, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second scalar, by element.</p> <p>The following instances are supported:</p> <pre>csipl_cvmsa_inter_f csipl_cvmsa_inter_d</pre>
<pre>void csipl_cvmsa_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_cscalar_P b_re, csipl_cscalar_P b_im, csipl_cscalar_P c_re, csipl_cscalar_P c_im, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second scalar, by element.</p> <p>The following instances are supported:</p> <pre>csipl_cvmsa_split_f csipl_cvmsa_split_d</pre>

6.5 Logical Operations

Prototype	Description
<pre>csipl_scalar_bl csipl_valltrue_bl(signed int * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns true if all the elements of a vector are true.</p>
<pre>csipl_scalar_bl csipl_malltrue_bl(signed int * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns true if all the elements of a vector are true.</p>

Prototype	Description
<pre> <i>csipl_scalar_bl</i> <i>csipl_vanytrue_bl</i>(signed int * A, <i>csipl_stride</i> strideA, <i>csipl_length</i> n); </pre>	Returns true if one or more elements of a vector are true.
<pre> <i>csipl_scalar_bl</i> <i>csipl_manytrue_bl</i>(signed int * A, <i>csipl_stride</i> strideA, <i>csipl_length</i> n); </pre>	Returns true if one or more elements of a vector are true.
<pre> void <i>csipl_vleq_P</i>(scalar_P * A, <i>csipl_stride</i> strideA, scalar_P * B, <i>csipl_stride</i> strideB, signed int * R, <i>csipl_stride</i> strideR, <i>csipl_length</i> n); </pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors. The following instances are supported:</p> <pre> <i>csipl_vleq_f</i> <i>csipl_vleq_d</i> <i>csipl_vleq_i</i> </pre>
<pre> void <i>csipl_cvleq_inter_P</i>(void * A, <i>csipl_stride</i> strideA, void * B, <i>csipl_stride</i> strideB, signed int * R, <i>csipl_stride</i> strideR, <i>csipl_length</i> n); </pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors. The following instances are supported:</p> <pre> <i>csipl_cvleq_inter_f</i> <i>csipl_cvleq_inter_d</i> <i>csipl_cvleq_inter_i</i> </pre>
<pre> void <i>csipl_cvleq_split_P</i>(scalar_P * A_re, scalar_P * A_im, <i>csipl_stride</i> strideA, scalar_P * B_re, scalar_P * B_im, <i>csipl_stride</i> strideB, signed int * R_re, signed int * R_im, <i>csipl_stride</i> strideR, <i>csipl_length</i> n); </pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors. The following instances are supported:</p> <pre> <i>csipl_cvleq_split_f</i> <i>csipl_cvleq_split_d</i> <i>csipl_cvleq_split_i</i> </pre>
<pre> void <i>csipl_mleq_P</i>(scalar_P * A, int ldA, scalar_P * B, int ldB, signed int * R, <i>csipl_stride</i> strideR, <i>csipl_length</i> m, <i>csipl_length</i> n); </pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors/matrices. The following instances are supported:</p> <pre> <i>csipl_mleq_f</i> <i>csipl_mleq_d</i> <i>csipl_mleq_i</i> </pre>
<pre> void <i>csipl_cmleq_inter_P</i>(void * A, int ldA, void * B, int ldB, signed int * R, <i>csipl_stride</i> strideR, <i>csipl_length</i> m, <i>csipl_length</i> n); </pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors/matrices. The following instances are supported:</p> <pre> <i>csipl_cmleq_inter_f</i> <i>csipl_cmleq_inter_d</i> <i>csipl_cmleq_inter_i</i> </pre>

Prototype	Description
<pre>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);</pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_cmleq_split_f csipl_cmleq_split_d csipl_cmleq_split_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'greater than or equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vlge_f csipl_vlge_d csipl_vlge_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'greater than or equal', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mlge_f csipl_mlge_d csipl_mlge_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'greater than', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vlgt_f csipl_vlgt_d csipl_vlgt_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'greater than', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mlgt_f csipl_mlgt_d csipl_mlgt_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'less than or equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vlle_f csipl_vlle_d csipl_vlle_i</pre>

Prototype	Description
<pre>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);</pre>	<p>Computes the boolean comparison of 'less than or equal', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mlle_f csipl_mlle_d csipl_mlle_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'less than', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vllt_f csipl_vllt_d csipl_vllt_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'less than', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mllt_f csipl_mllt_d csipl_mllt_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'not equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vlne_f csipl_vlne_d csipl_vlne_i</pre>
<pre>void csipl_cvlne_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of 'not equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_cvlne_inter_f csipl_cvlne_inter_d csipl_cvlne_inter_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of 'not equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_cvlne_split_f csipl_cvlne_split_d csipl_cvlne_split_i</pre>

Prototype	Description
<pre>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);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mlne_f csipl_mlne_d csipl_mlne_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_cmlne_inter_f csipl_cmlne_inter_d csipl_cmlne_inter_i</pre>
<pre>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);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>csipl_cmlne_split_f csipl_cmlne_split_d csipl_cmlne_split_i</pre>

6.6 Selection Operations

Prototype	Description
<pre>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);</pre>	<p>Computes the generalised double clip, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vclip_f csipl_vclip_d csipl_vclip_i csipl_vclip_si</pre>
<pre>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);</pre>	<p>Computes the generalised inverted double clip, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vinvclip_f csipl_vinvclip_d csipl_vinvclip_i csipl_vinvclip_si</pre>

Prototype	Description
<pre>unsigned int csipl_vindexbool(signed int * X, csipl_stride strideX, unsigned int * Y, csipl_stride strideY, csipl_length n);</pre>	<p>Computes an index vector of the indices of the non-false elements of the boolean vector, and returns the number of non-false elements.</p>
<pre>void csipl_vmax_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum, by element, of two vectors. The following instances are supported:</p> <pre>csipl_vmax_f csipl_vmax_d</pre>
<pre>void csipl_vmaxmg_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum magnitude (absolute value), by element, of two vectors. The following instances are supported:</p> <pre>csipl_vmaxmg_f csipl_vmaxmg_d</pre>
<pre>void csipl_vcmaxmgsq_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum magnitude squared, by element, of two complex vectors. The following instances are supported:</p> <pre>csipl_vcmaxmgsq_inter_f csipl_vcmaxmgsq_inter_d</pre>
<pre>void csipl_vcmaxmgsq_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, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum magnitude squared, by element, of two complex vectors. The following instances are supported:</p> <pre>csipl_vcmaxmgsq_split_f csipl_vcmaxmgsq_split_d</pre>
<pre>scalar_P csipl_vcmaxmgsqval_inter_P(void * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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. The following instances are supported:</p> <pre>csipl_vcmaxmgsqval_inter_f csipl_vcmaxmgsqval_inter_d</pre>

Prototype	Description
<pre>scalar_P csipl_vcmaxmgsqval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <pre>csipl_vcmaxmgsqval_split_f csipl_vcmaxmgsqval_split_d</pre>
<pre>scalar_P csipl_vmaxmgval_P(scalar_P * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <pre>csipl_vmaxmgval_f csipl_vmaxmgval_d</pre>
<pre>scalar_P csipl_vmaxval_P(scalar_P * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <pre>csipl_vmaxval_f csipl_vmaxval_d</pre>
<pre>void csipl_vmin_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the minimum, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vmin_f csipl_vmin_d</pre>
<pre>void csipl_vminmg_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the minimum magnitude (absolute value), by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vminmg_f csipl_vminmg_d</pre>
<pre>void csipl_vcminmgsq_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the minimum magnitude squared, by element, of two complex vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vcminmgsq_inter_f csipl_vcminmgsq_inter_d</pre>

Prototype	Description
<pre>void csipl_vcminmgsq_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, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the minimum magnitude squared, by element, of two complex vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vcminmgsq_split_f</code> <code>csipl_vcminmgsq_split_d</code></p>
<pre>scalar_P csipl_vcminmgsqval_inter_P(void * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <p><code>csipl_vcminmgsqval_inter_f</code> <code>csipl_vcminmgsqval_inter_d</code></p>
<pre>scalar_P csipl_vcminmgsqval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <p><code>csipl_vcminmgsqval_split_f</code> <code>csipl_vcminmgsqval_split_d</code></p>
<pre>scalar_P csipl_vminmgval_P(scalar_P * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <p><code>csipl_vminmgval_f</code> <code>csipl_vminmgval_d</code></p>
<pre>scalar_P csipl_vminval_P(scalar_P * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <p><code>csipl_vminval_f</code> <code>csipl_vminval_d</code></p>

6.7 Bitwise and Boolean Logical Operators

Prototype	Description
<pre>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);</pre>	<p>Computes the bitwise and, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vand_i</code> <code>csipl_vand_si</code> <code>csipl_vand_bl</code></p>

Prototype	Description
<pre>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);</pre>	<p>Computes the bitwise and, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mand_i csipl_mand_si csipl_mand_bl</pre>
<pre>void csipl_vnot_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the bitwise not (one's complement), by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vnot_i csipl_vnot_si csipl_vnot_bl</pre>
<pre>void csipl_mnot_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the bitwise not (one's complement), by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mnot_i csipl_mnot_si csipl_mnot_bl</pre>
<pre>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);</pre>	<p>Computes the bitwise inclusive or, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vor_i csipl_vor_si csipl_vor_bl</pre>
<pre>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);</pre>	<p>Computes the bitwise inclusive or, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mor_i csipl_mor_si csipl_mor_bl</pre>
<pre>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);</pre>	<p>Computes the bitwise exclusive or, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_mxor_i csipl_mxor_si csipl_mxor_bl</pre>

6.8 Element Generation and Copy

Prototype	Description
<pre>void csipl_vcopy_P_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_vcopy_f_f csipl_vcopy_f_d csipl_vcopy_f_i csipl_vcopy_f_si csipl_vcopy_f_bl csipl_vcopy_d_f csipl_vcopy_d_d csipl_vcopy_d_i csipl_vcopy_d_si csipl_vcopy_d_bl csipl_vcopy_i_f csipl_vcopy_i_d csipl_vcopy_i_i csipl_vcopy_i_si csipl_vcopy_i_vi csipl_vcopy_si_f csipl_vcopy_si_d csipl_vcopy_si_i csipl_vcopy_si_si csipl_vcopy_bl_f csipl_vcopy_bl_d csipl_vcopy_bl_bl csipl_vcopy_vi_i csipl_vcopy_vi_vi csipl_vcopy_mi_mi</pre>
<pre>void csipl_cvcopy_inter_P_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_cvcopy_inter_f_f csipl_cvcopy_inter_f_d csipl_cvcopy_inter_d_f csipl_cvcopy_inter_d_d</pre>
<pre>void csipl_cvcopy_split_P_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);</pre>	<p>Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_cvcopy_split_f_f csipl_cvcopy_split_f_d csipl_cvcopy_split_d_f csipl_cvcopy_split_d_d</pre>

Prototype	Description
<pre>void csipl_mcopy_P_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_mcopy_f_f csipl_mcopy_f_d csipl_mcopy_f_i csipl_mcopy_f_si csipl_mcopy_f_bl csipl_mcopy_d_f csipl_mcopy_d_d csipl_mcopy_d_i csipl_mcopy_d_si csipl_mcopy_d_bl csipl_mcopy_i_f csipl_mcopy_i_d csipl_mcopy_i_i csipl_mcopy_i_si csipl_mcopy_si_f csipl_mcopy_si_d csipl_mcopy_si_i csipl_mcopy_si_si csipl_mcopy_bl_f csipl_mcopy_bl_d csipl_mcopy_bl_bl</pre>
<pre>void csipl_cmcopy_inter_P_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_cmcopy_inter_f_f csipl_cmcopy_inter_f_d csipl_cmcopy_inter_d_f csipl_cmcopy_inter_d_d</pre>
<pre>void csipl_cmcopy_split_P_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);</pre>	<p>Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_cmcopy_split_f_f csipl_cmcopy_split_f_d csipl_cmcopy_split_d_f csipl_cmcopy_split_d_d</pre>
<pre>void csipl_vfill_P(scalar_P a, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Fill a vector with a constant value.</p> <p>The following instances are supported:</p> <pre>csipl_vfill_f csipl_vfill_d csipl_vfill_i csipl_vfill_si</pre>
<pre>void csipl_cvfill_inter_P(csipl_cscalar_P a, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Fill a vector with a constant value.</p> <p>The following instances are supported:</p> <pre>csipl_cvfill_inter_f csipl_cvfill_inter_d</pre>

Prototype	Description
<pre>void csipl_cvfill_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Fill a vector with a constant value. The following instances are supported:</p> <pre>csipl_cvfill_split_f csipl_cvfill_split_d</pre>
<pre>void csipl_mfill_P(scalar_P a, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Fill a matrix with a constant value. The following instances are supported:</p> <pre>csipl_mfill_f csipl_mfill_d csipl_mfill_i csipl_mfill_si</pre>
<pre>void csipl_cmfill_inter_P(csipl_cscalar_P a, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Fill a matrix with a constant value. The following instances are supported:</p> <pre>csipl_cmfill_inter_f csipl_cmfill_inter_d</pre>
<pre>void csipl_cmfill_split_P(csipl_cscalar_P a_re, csipl_cscalar_P a_im, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Fill a matrix with a constant value. The following instances are supported:</p> <pre>csipl_cmfill_split_f csipl_cmfill_split_d</pre>
<pre>void csipl_vramp_P(scalar_P alpha, scalar_P beta, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes a vector ramp by starting at an initial value and incrementing each successive element by the ramp step size. The following instances are supported:</p> <pre>csipl_vramp_f csipl_vramp_d csipl_vramp_i csipl_vramp_si</pre>

6.9 Manipulation Operations

Prototype	Description
<pre>void csipl_vcplx_inter_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Form a complex vector from two real vectors. The following instances are supported:</p> <pre>csipl_vcplx_inter_f csipl_vcplx_inter_d</pre>

Prototype	Description
<pre>void csipl_vcmlpx_split_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Form a complex vector from two real vectors.</p> <p>The following instances are supported:</p> <p>csipl_vcmlpx_split_f csipl_vcmlpx_split_d</p>
<pre>void csipl_mcmlpx_P(scalar_P * A, int ldA, scalar_P * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Form a complex matrix from two real matrices.</p> <p>The following instances are supported:</p> <p>csipl_mcmlpx_f csipl_mcmlpx_d</p>
<pre>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);</pre>	<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.</p> <p>The following instances are supported:</p> <p>csipl_vgather_f csipl_vgather_d csipl_vgather_i csipl_vgather_si</p>
<pre>void csipl_cvgather_inter_P(void * X, csipl_stride strideX, unsigned int * I, csipl_stride strideI, void * Y, csipl_stride strideY, csipl_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <p>csipl_cvgather_inter_f csipl_cvgather_inter_d</p>
<pre>void csipl_cvgather_split_P(scalar_P * X_re, scalar_P * X_im, csipl_stride strideX, unsigned int * I_re, unsigned int * I_im, csipl_stride strideI, scalar_P * Y_re, scalar_P * Y_im, csipl_stride strideY, csipl_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <p>csipl_cvgather_split_f csipl_cvgather_split_d</p>

Prototype	Description
<pre>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);</pre>	<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.</p> <p>The following instances are supported:</p> <pre>csipl_mgather_f csipl_mgather_d csipl_mgather_i csipl_mgather_si</pre>
<pre>void csipl_cmgather_inter_P(void * X, int ldX, csipl_scalar_mi * I, csipl_stride strideI, void * Y, csipl_stride strideY, csipl_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <pre>csipl_cmgather_inter_f csipl_cmgather_inter_d</pre>
<pre>void csipl_cmgather_split_P(scalar_P * X_re, scalar_P * X_im, int ldX, csipl_scalar_mi * I_re, csipl_scalar_mi * I_im, csipl_stride strideI, scalar_P * Y_re, scalar_P * Y_im, csipl_stride strideY, csipl_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <pre>csipl_cmgather_split_f csipl_cmgather_split_d</pre>
<pre>void csipl_vimag_inter_P(void * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Extract the imaginary part of a complex vector.</p> <p>The following instances are supported:</p> <pre>csipl_vimag_inter_f csipl_vimag_inter_d</pre>
<pre>void csipl_vimag_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Extract the imaginary part of a complex vector.</p> <p>The following instances are supported:</p> <pre>csipl_vimag_split_f csipl_vimag_split_d</pre>
<pre>void csipl_mimag_P(void * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Extract the imaginary part of a complex matrix.</p> <p>The following instances are supported:</p> <pre>csipl_mimag_f csipl_mimag_d</pre>

Prototype	Description
<pre>void csipl_vpolar_inter_P(void * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, scalar_P * P, csipl_stride strideP, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <p><code>csipl_vpolar_inter_f</code> <code>csipl_vpolar_inter_d</code></p>
<pre>void csipl_vpolar_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * R, csipl_stride strideR, scalar_P * P, csipl_stride strideP, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <p><code>csipl_vpolar_split_f</code> <code>csipl_vpolar_split_d</code></p>
<pre>void csipl_mpolar_P(void * A, int ldA, scalar_P * R, int ldR, scalar_P * P, int ldP, csipl_length m, csipl_length n);</pre>	<p>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.</p> <p>The following instances are supported:</p> <p><code>csipl_mpolar_f</code> <code>csipl_mpolar_d</code></p>
<pre>void csipl_vreal_inter_P(void * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Extract the real part of a complex vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vreal_inter_f</code> <code>csipl_vreal_inter_d</code></p>
<pre>void csipl_vreal_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Extract the real part of a complex vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vreal_split_f</code> <code>csipl_vreal_split_d</code></p>
<pre>void csipl_mreal_P(void * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Extract the real part of a complex matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_mreal_f</code> <code>csipl_mreal_d</code></p>

Prototype	Description
<pre>void csipl_vrect_inter_P(scalar_P * R, csipl_stride strideR, scalar_P * P, csipl_stride strideP, void * A, csipl_stride strideA, csipl_length n);</pre>	<p>Convert a pair of real vectors from complex polar to complex rectangular form. The following instances are supported:</p> <p><code>csipl_vrect_inter_f</code> <code>csipl_vrect_inter_d</code></p>
<pre>void csipl_vrect_split_P(scalar_P * R, csipl_stride strideR, scalar_P * P, csipl_stride strideP, scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	<p>Convert a pair of real vectors from complex polar to complex rectangular form. The following instances are supported:</p> <p><code>csipl_vrect_split_f</code> <code>csipl_vrect_split_d</code></p>
<pre>void csipl_mrect_P(scalar_P * R, int ldR, scalar_P * P, int ldP, void * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Convert a pair of real matrices from complex polar to complex rectangular form. The following instances are supported:</p> <p><code>csipl_mrect_f</code> <code>csipl_mrect_d</code></p>
<pre>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);</pre>	<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. The following instances are supported:</p> <p><code>csipl_vscatter_f</code> <code>csipl_vscatter_d</code> <code>csipl_vscatter_i</code> <code>csipl_vscatter_si</code></p>
<pre>void csipl_cvscatter_inter_P(void * X, csipl_stride strideX, void * Y, csipl_stride strideY, unsigned int * I, csipl_stride strideI, csipl_length n);</pre>	<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. The following instances are supported:</p> <p><code>csipl_cvscatter_inter_f</code> <code>csipl_cvscatter_inter_d</code></p>

Prototype	Description
<pre>void csiplt_cvscatter_split_P(scalar_P * X_re, scalar_P * X_im, csiplt_stride strideX, scalar_P * Y_re, scalar_P * Y_im, csiplt_stride strideY, unsigned int * I_re, unsigned int * I_im, csiplt_stride strideI, csiplt_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <p>csiplt_cvscatter_split_f csiplt_cvscatter_split_d</p>
<pre>void csiplt_mscatter_P(scalar_P * X, csiplt_stride strideX, scalar_P * Y, int ldY, csiplt_scalar_mi * I, csiplt_stride strideI, csiplt_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <p>csiplt_mscatter_f csiplt_mscatter_d csiplt_mscatter_i csiplt_mscatter_si</p>
<pre>void csiplt_cmscatter_inter_P(void * X, csiplt_stride strideX, void * Y, int ldY, csiplt_scalar_mi * I, csiplt_stride strideI, csiplt_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <p>csiplt_cmscatter_inter_f csiplt_cmscatter_inter_d</p>
<pre>void csiplt_cmscatter_split_P(scalar_P * X_re, scalar_P * X_im, csiplt_stride strideX, scalar_P * Y_re, scalar_P * Y_im, int ldY, csiplt_scalar_mi * I_re, csiplt_scalar_mi * I_im, csiplt_stride strideI, csiplt_length n);</pre>	<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.</p> <p>The following instances are supported:</p> <p>csiplt_cmscatter_split_f csiplt_cmscatter_split_d</p>
<pre>void csiplt_cvswap_inter_P(void * A, csiplt_stride strideA, void * B, csiplt_stride strideB, csiplt_length n);</pre>	<p>Swap elements between two vectors.</p> <p>The following instances are supported:</p> <p>csiplt_cvswap_inter_f csiplt_cvswap_inter_d</p>

Prototype	Description
<pre>void csipl_cvswap_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, csipl_length n);</pre>	<p>Swap elements between two vectors. The following instances are supported:</p> <pre>csipl_cvswap_split_f csipl_cvswap_split_d</pre>
<pre>void csipl_mswap_P(scalar_P * A, int ldA, scalar_P * B, int ldB, csipl_length m, csipl_length n);</pre>	<p>Swap elements between two matrices. The following instances are supported:</p> <pre>csipl_mswap_f csipl_mswap_d csipl_mswap_i csipl_mswap_si</pre>
<pre>void csipl_cmswap_inter_P(void * A, int ldA, void * B, int ldB, csipl_length m, csipl_length n);</pre>	<p>Swap elements between two matrices. The following instances are supported:</p> <pre>csipl_cmswap_inter_f csipl_cmswap_inter_d</pre>
<pre>void csipl_cmswap_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B_re, scalar_P * B_im, int ldB, csipl_length m, csipl_length n);</pre>	<p>Swap elements between two matrices. The following instances are supported:</p> <pre>csipl_cmswap_split_f csipl_cmswap_split_d</pre>

Chapter 7. Signal Processing Functions

7.1 FFT Functions

Prototype	Description
<pre><i>csipl_fft_P</i> * csipl_ccfftip_create_P(<i>csipl_index</i> length, <i>scalar_P</i> scale, <i>csipl_fft_dir</i> dir, unsigned int ntimes, <i>csipl_alg_hint</i> hint);</pre>	Create a 1D FFT object. The following instances are supported: <code>csipl_ccfftip_create_f</code> <code>csipl_ccfftip_create_d</code>
<pre><i>csipl_fft_P</i> * csipl_ccfftop_create_P(<i>csipl_index</i> length, <i>scalar_P</i> scale, <i>csipl_fft_dir</i> dir, unsigned int ntimes, <i>csipl_alg_hint</i> hint);</pre>	Create a 1D FFT object. The following instances are supported: <code>csipl_ccfftop_create_f</code> <code>csipl_ccfftop_create_d</code>
<pre><i>csipl_fft_P</i> * csipl_crfftop_create_P(<i>csipl_index</i> length, <i>scalar_P</i> scale, unsigned int ntimes, <i>csipl_alg_hint</i> hint);</pre>	Create a 1D FFT object. The following instances are supported: <code>csipl_crfftop_create_f</code> <code>csipl_crfftop_create_d</code>
<pre><i>csipl_fft_P</i> * csipl_rcfftop_create_P(<i>csipl_index</i> length, <i>scalar_P</i> scale, unsigned int ntimes, <i>csipl_alg_hint</i> hint);</pre>	Create a 1D FFT object. The following instances are supported: <code>csipl_rcfftop_create_f</code> <code>csipl_rcfftop_create_d</code>
<pre>void csipl_ccfftip_inter_P(<i>csipl_fft_P</i> * plan, void * xy, <i>csipl_stride</i> stridexy);</pre>	Apply a complex-to-complex Fast Fourier Transform (FFT). The following instances are supported: <code>csipl_ccfftip_inter_f</code> <code>csipl_ccfftip_inter_d</code>
<pre>void csipl_ccfftip_split_P(<i>csipl_fft_P</i> * plan, <i>scalar_P</i> * xy_re, <i>scalar_P</i> * xy_im, <i>csipl_stride</i> stridexy);</pre>	Apply a complex-to-complex Fast Fourier Transform (FFT). The following instances are supported: <code>csipl_ccfftip_split_f</code> <code>csipl_ccfftip_split_d</code>
<pre>void csipl_ccfftop_inter_P(<i>csipl_fft_P</i> * plan, void * x, <i>csipl_stride</i> stridex, void * y, <i>csipl_stride</i> stridey);</pre>	Apply a complex-to-complex Fast Fourier Transform (FFT). The following instances are supported: <code>csipl_ccfftop_inter_f</code> <code>csipl_ccfftop_inter_d</code>

Prototype	Description
<pre>void csipl_ccffftop_split_P(csipl_fft_P * plan, scalar_P * x_re, scalar_P * x_im, csipl_stride stridex, scalar_P * y_re, scalar_P * y_im, csipl_stride stridey);</pre>	<p>Apply a complex-to-complex Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_ccffftop_split_f</code> <code>csipl_ccffftop_split_d</code></p>
<pre>void csipl_crffftop_inter_P(csipl_fft_P * plan, void * x, csipl_stride stridex, scalar_P * y, csipl_stride stridey);</pre>	<p>Apply a complex-to-real Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_crffftop_inter_f</code> <code>csipl_crffftop_inter_d</code></p>
<pre>void csipl_crffftop_split_P(csipl_fft_P * plan, scalar_P * x_re, scalar_P * x_im, csipl_stride stridex, scalar_P * y, csipl_stride stridey);</pre>	<p>Apply a complex-to-real Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_crffftop_split_f</code> <code>csipl_crffftop_split_d</code></p>
<pre>void csipl_rcffftop_inter_P(csipl_fft_P * plan, scalar_P * x, csipl_stride stridex, void * y, csipl_stride stridey);</pre>	<p>Apply a real-to-complex Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_rcffftop_inter_f</code> <code>csipl_rcffftop_inter_d</code></p>
<pre>void csipl_rcffftop_split_P(csipl_fft_P * plan, scalar_P * x, csipl_stride stridex, scalar_P * y_re, scalar_P * y_im, csipl_stride stridey);</pre>	<p>Apply a real-to-complex Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_rcffftop_split_f</code> <code>csipl_rcffftop_split_d</code></p>
<pre>int csipl_fft_destroy_P(csipl_fft_P * plan);</pre>	<p>Destroy an FFT object. The following instances are supported:</p> <p><code>csipl_fft_destroy_f</code> <code>csipl_fft_destroy_d</code></p>
<pre>void csipl_fft_getattr_P(csipl_fft_P * plan, csipl_fft_attr_P * attr);</pre>	<p>Return the attributes of an FFT object. The following instances are supported:</p> <p><code>csipl_fft_getattr_f</code> <code>csipl_fft_getattr_d</code></p>
<pre>csipl_fftm_P * csipl_ccfftmop_create_P(csipl_index rows, csipl_index cols, scalar_P scale, csipl_fft_dir dir, csipl_major major, unsigned int ntimes, csipl_alg_hint hint);</pre>	<p>Create a 1D multiple FFT object. The following instances are supported:</p> <p><code>csipl_ccfftmop_create_f</code> <code>csipl_ccfftmop_create_d</code></p>

Prototype	Description
<pre> <i>csipl_fftm_P</i> * <i>csipl_ccfftmip_create_P</i>(<i>csipl_index</i> rows, <i>csipl_index</i> cols, <i>scalar_P</i> scale, <i>csipl_fft_dir</i> dir, <i>csipl_major</i> major, <i>unsigned int</i> ntimes, <i>csipl_alg_hint</i> hint); </pre>	<p>Create a 1D multiple FFT object. The following instances are supported:</p> <pre> <i>csipl_ccfftmip_create_f</i> <i>csipl_ccfftmip_create_d</i> </pre>
<pre> <i>csipl_fftm_P</i> * <i>csipl_crfftmop_create_P</i>(<i>csipl_index</i> rows, <i>csipl_index</i> cols, <i>scalar_P</i> scale, <i>csipl_major</i> major, <i>unsigned int</i> ntimes, <i>csipl_alg_hint</i> hint); </pre>	<p>Create a 1D multiple FFT object. The following instances are supported:</p> <pre> <i>csipl_crfftmop_create_f</i> <i>csipl_crfftmop_create_d</i> </pre>
<pre> <i>csipl_fftm_P</i> * <i>csipl_rcfftmop_create_P</i>(<i>csipl_index</i> rows, <i>csipl_index</i> cols, <i>scalar_P</i> scale, <i>csipl_major</i> major, <i>unsigned int</i> ntimes, <i>csipl_alg_hint</i> hint); </pre>	<p>Create a 1D multiple FFT object. The following instances are supported:</p> <pre> <i>csipl_rcfftmop_create_f</i> <i>csipl_rcfftmop_create_d</i> </pre>
<pre> <i>int</i> <i>csipl_fftm_destroy_P</i>(<i>csipl_fftm_P</i> * <i>plan</i>); </pre>	<p>Destroy an FFT object. The following instances are supported:</p> <pre> <i>csipl_fftm_destroy_f</i> <i>csipl_fftm_destroy_d</i> </pre>
<pre> <i>void</i> <i>csipl_fftm_getattr_P</i>(<i>csipl_fftm_P</i> * <i>plan</i>, <i>csipl_fftm_attr_P</i> * <i>attr</i>); </pre>	<p>Return the attributes of an FFT object. The following instances are supported:</p> <pre> <i>csipl_fftm_getattr_f</i> <i>csipl_fftm_getattr_d</i> </pre>
<pre> <i>void</i> <i>csipl_ccfftmip_inter_P</i>(<i>csipl_fftm_P</i> * <i>plan</i>, <i>void</i> * <i>XY</i>, <i>int</i> <i>ldXY</i>); </pre>	<p>Apply a multiple complex-to-complex Fast Fourier Transform (FFT). The following instances are supported:</p> <pre> <i>csipl_ccfftmip_inter_f</i> <i>csipl_ccfftmip_inter_d</i> </pre>
<pre> <i>void</i> <i>csipl_ccfftmip_split_P</i>(<i>csipl_fftm_P</i> * <i>plan</i>, <i>scalar_P</i> * <i>XY_re</i>, <i>scalar_P</i> * <i>XY_im</i>, <i>int</i> <i>ldXY</i>); </pre>	<p>Apply a multiple complex-to-complex Fast Fourier Transform (FFT). The following instances are supported:</p> <pre> <i>csipl_ccfftmip_split_f</i> <i>csipl_ccfftmip_split_d</i> </pre>
<pre> <i>void</i> <i>csipl_ccfftmop_inter_P</i>(<i>csipl_fftm_P</i> * <i>plan</i>, <i>void</i> * <i>X</i>, <i>int</i> <i>ldX</i>, <i>void</i> * <i>Y</i>, <i>int</i> <i>ldY</i>); </pre>	<p>Apply a multiple complex-to-complex Fast Fourier Transform (FFT). The following instances are supported:</p> <pre> <i>csipl_ccfftmop_inter_f</i> <i>csipl_ccfftmop_inter_d</i> </pre>

Prototype	Description
<pre>void csipl_ccfftmop_split_P(csipl_fftm_P * plan, scalar_P * X_re, scalar_P * X_im, int ldX, scalar_P * Y_re, scalar_P * Y_im, int ldY);</pre>	<p>Apply a multiple complex-to-complex Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_ccfftmop_split_f</code> <code>csipl_ccfftmop_split_d</code></p>
<pre>void csipl_crfftmop_inter_P(csipl_fftm_P * plan, void * X, int ldX, scalar_P * Y, int ldY);</pre>	<p>Apply a multiple complex-to-real Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_crfftmop_inter_f</code> <code>csipl_crfftmop_inter_d</code></p>
<pre>void csipl_crfftmop_split_P(csipl_fftm_P * plan, scalar_P * X_re, scalar_P * X_im, int ldX, scalar_P * Y, int ldY);</pre>	<p>Apply a multiple complex-to-real Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_crfftmop_split_f</code> <code>csipl_crfftmop_split_d</code></p>
<pre>void csipl_rcfftmop_inter_P(csipl_fftm_P * plan, scalar_P * X, int ldX, void * Y, int ldY);</pre>	<p>Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_rcfftmop_inter_f</code> <code>csipl_rcfftmop_inter_d</code></p>
<pre>void csipl_rcfftmop_split_P(csipl_fftm_P * plan, scalar_P * X, int ldX, scalar_P * Y_re, scalar_P * Y_im, int ldY);</pre>	<p>Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_rcfftmop_split_f</code> <code>csipl_rcfftmop_split_d</code></p>
<pre>csipl_fft2d_P * csipl_ccfft2dop_create_P(csipl_index rows, csipl_index cols, scalar_P scale, csipl_fft_dir dir, unsigned int ntimes, csipl_alg_hint hint);</pre>	<p>Create a 2D FFT object. The following instances are supported:</p> <p><code>csipl_ccfft2dop_create_f</code> <code>csipl_ccfft2dop_create_d</code></p>
<pre>csipl_fft2d_P * csipl_ccfft2dip_create_P(csipl_index rows, csipl_index cols, scalar_P scale, csipl_fft_dir dir, unsigned int ntimes, csipl_alg_hint hint);</pre>	<p>Create a 2D FFT object. The following instances are supported:</p> <p><code>csipl_ccfft2dip_create_f</code> <code>csipl_ccfft2dip_create_d</code></p>
<pre>csipl_fft2d_P * csipl_crfft2dop_create_P(csipl_index rows, csipl_index cols, scalar_P scale, unsigned int ntimes, csipl_alg_hint hint);</pre>	<p>Create a 2D FFT object. The following instances are supported:</p> <p><code>csipl_crfft2dop_create_f</code> <code>csipl_crfft2dop_create_d</code></p>

Prototype	Description
<pre> csipl_fft2d_P * csipl_rcfft2dop_create_P(csipl_index rows, csipl_index cols, scalar_P scale, unsigned int ntimes, csipl_alg_hint hint); </pre>	<p>Create a 2D FFT object. The following instances are supported:</p> <p>csipl_rcfft2dop_create_f csipl_rcfft2dop_create_d</p>
<pre> void csipl_ccfft2dip_inter_P(csipl_fft2d_P * plan, void * XY, int ldXY); </pre>	<p>Apply a complex-to-complex 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p>csipl_ccfft2dip_inter_f csipl_ccfft2dip_inter_d</p>
<pre> void csipl_ccfft2dip_split_P(csipl_fft2d_P * plan, scalar_P * XY_re, scalar_P * XY_im, int ldXY); </pre>	<p>Apply a complex-to-complex 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p>csipl_ccfft2dip_split_f csipl_ccfft2dip_split_d</p>
<pre> void csipl_ccfft2dop_inter_P(csipl_fft2d_P * plan, void * X, int ldX, void * Y, int ldY); </pre>	<p>Apply a complex-to-complex 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p>csipl_ccfft2dop_inter_f csipl_ccfft2dop_inter_d</p>
<pre> void csipl_ccfft2dop_split_P(csipl_fft2d_P * plan, scalar_P * X_re, scalar_P * X_im, int ldX, scalar_P * Y_re, scalar_P * Y_im, int ldY); </pre>	<p>Apply a complex-to-complex 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p>csipl_ccfft2dop_split_f csipl_ccfft2dop_split_d</p>
<pre> void csipl_crfft2dop_inter_P(csipl_fft2d_P * plan, void * X, int ldX, scalar_P * Y, int ldY); </pre>	<p>Apply a complex-to-real 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p>csipl_crfft2dop_inter_f csipl_crfft2dop_inter_d</p>
<pre> void csipl_crfft2dop_split_P(csipl_fft2d_P * plan, scalar_P * X_re, scalar_P * X_im, int ldX, scalar_P * Y, int ldY); </pre>	<p>Apply a complex-to-real 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p>csipl_crfft2dop_split_f csipl_crfft2dop_split_d</p>
<pre> void csipl_rcfft2dop_inter_P(csipl_fft2d_P * plan, scalar_P * X, int ldX, void * Y, int ldY); </pre>	<p>Apply a real-to-complex 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p>csipl_rcfft2dop_inter_f csipl_rcfft2dop_inter_d</p>

Prototype	Description
<pre>void csipl_rcfft2dop_split_P(csipl_fft2d_P * plan, scalar_P * X, int ldX, scalar_P * Y_re, scalar_P * Y_im, int ldY);</pre>	<p>Apply a real-to-complex 2D Fast Fourier Transform (FFT). The following instances are supported:</p> <p><code>csipl_rcfft2dop_split_f</code> <code>csipl_rcfft2dop_split_d</code></p>
<pre>int csipl_fft2d_destroy_P(csipl_fft2d_P * plan);</pre>	<p>Destroy an FFT object. The following instances are supported:</p> <p><code>csipl_fft2d_destroy_f</code> <code>csipl_fft2d_destroy_d</code></p>
<pre>void csipl_fft2d_getattr_P(csipl_fft2d_P * plan, csipl_fft2d_attr_P * attr);</pre>	<p>Return the attributes of an FFT object. The following instances are supported:</p> <p><code>csipl_fft2d_getattr_f</code> <code>csipl_fft2d_getattr_d</code></p>

7.2 Convolution/Correlation Functions

Prototype	Description
<pre>csipl_conv1d_P * csipl_conv1d_create_P(scalar_P * 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);</pre>	<p>Create a decimated 1D convolution filter object. The following instances are supported:</p> <p><code>csipl_conv1d_create_f</code> <code>csipl_conv1d_create_d</code></p>
<pre>int csipl_conv1d_destroy_P(csipl_conv1d_P * plan);</pre>	<p>Destroy a 1D convolution object. The following instances are supported:</p> <p><code>csipl_conv1d_destroy_f</code> <code>csipl_conv1d_destroy_d</code></p>
<pre>void csipl_conv1d_getattr_P(csipl_conv1d_P * plan, csipl_conv1d_attr_P * attr);</pre>	<p>Returns the attributes for a 1D convolution object. The following instances are supported:</p> <p><code>csipl_conv1d_getattr_f</code> <code>csipl_conv1d_getattr_d</code></p>
<pre>void csipl_convolve1d_P(csipl_conv1d_P * plan, scalar_P * x, csipl_stride stridex, scalar_P * y, csipl_stride stridey, csipl_length n);</pre>	<p>Compute a decimated real one-dimensional (1D) convolution of two vectors. The following instances are supported:</p> <p><code>csipl_convolve1d_f</code> <code>csipl_convolve1d_d</code></p>

Prototype	Description
<pre> <i>csipl_conv2d_P</i> * <i>csipl_conv2d_create_P</i>(scalar_P * H, int ldH, <i>csipl_symmetry</i> symm, unsigned int P, unsigned int Q, unsigned int D, <i>csipl_support_region</i> support, unsigned int ntimes, <i>csipl_alg_hint</i> hint, <i>csipl_length</i> m, <i>csipl_length</i> n); </pre>	<p>Create a decimated 2D convolution filter object. The following instances are supported:</p> <pre> <i>csipl_conv2d_create_f</i> <i>csipl_conv2d_create_d</i> </pre>
<pre> int <i>csipl_conv2d_destroy_P</i>(<i>csipl_conv2d_P</i> * plan); </pre>	<p>Destroy a 2D convolution object. The following instances are supported:</p> <pre> <i>csipl_conv2d_destroy_f</i> <i>csipl_conv2d_destroy_d</i> </pre>
<pre> void <i>csipl_conv2d_getattr_P</i>(<i>csipl_conv2d_P</i> * plan, <i>csipl_conv2d_attr_P</i> * attr); </pre>	<p>Returns the attributes for a 2D convolution object. The following instances are supported:</p> <pre> <i>csipl_conv2d_getattr_f</i> <i>csipl_conv2d_getattr_d</i> </pre>
<pre> void <i>csipl_convolve2d_P</i>(<i>csipl_conv2d_P</i> * plan, scalar_P * x, int ldx, scalar_P * y, int ldy, <i>csipl_length</i> m, <i>csipl_length</i> n); </pre>	<p>Compute a decimated real two-dimensional (2D) convolution of two matrices. The following instances are supported:</p> <pre> <i>csipl_convolve2d_f</i> <i>csipl_convolve2d_d</i> </pre>
<pre> <i>csipl_Dcorr1d_P</i> * <i>csipl_Dcorr1d_create_P</i>(unsigned int M, unsigned int N, <i>csipl_support_region</i> support, unsigned int ntimes, <i>csipl_alg_hint</i> hint); </pre>	<p>Create a 1D correlation object. The following instances are supported:</p> <pre> <i>csipl_corr1d_create_f</i> <i>csipl_corr1d_create_d</i> <i>csipl_ccorr1d_create_f</i> <i>csipl_ccorr1d_create_d</i> </pre>
<pre> int <i>csipl_Dcorr1d_destroy_P</i>(<i>csipl_Dcorr1d_P</i> * plan); </pre>	<p>Destroy a 1D correlation object. The following instances are supported:</p> <pre> <i>csipl_corr1d_destroy_f</i> <i>csipl_corr1d_destroy_d</i> <i>csipl_ccorr1d_destroy_f</i> <i>csipl_ccorr1d_destroy_d</i> </pre>
<pre> void <i>csipl_Dcorr1d_getattr_P</i>(<i>csipl_Dcorr1d_P</i> * plan, <i>csipl_Dcorr1d_attr_P</i> * attr); </pre>	<p>Return the attributes for a 1D correlation object. The following instances are supported:</p> <pre> <i>csipl_corr1d_getattr_f</i> <i>csipl_corr1d_getattr_d</i> <i>csipl_ccorr1d_getattr_f</i> <i>csipl_ccorr1d_getattr_d</i> </pre>

Prototype	Description
<pre>void csipl_correlate1d_P(csipl_corr1d_P * plan, csipl_bias bias, scalar_P * ref, csipl_stride strideref, scalar_P * x, csipl_stride stridex, scalar_P * y, csipl_stride stridey, csipl_length n);</pre>	<p>Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported:</p> <p><code>csipl_correlate1d_f</code> <code>csipl_correlate1d_d</code></p>
<pre>void csipl_ccorrelate1d_inter_P(csipl_ccorr1d_P * plan, csipl_bias bias, void * ref, csipl_stride strideref, void * x, csipl_stride stridex, void * y, csipl_stride stridey, csipl_length n);</pre>	<p>Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported:</p> <p><code>csipl_ccorrelate1d_inter_f</code> <code>csipl_ccorrelate1d_inter_d</code></p>
<pre>void csipl_ccorrelate1d_split_P(csipl_ccorr1d_P * plan, csipl_bias bias, scalar_P * ref_re, scalar_P * ref_im, csipl_stride strideref, scalar_P * x_re, scalar_P * x_im, csipl_stride stridex, scalar_P * y_re, scalar_P * y_im, csipl_stride stridey, csipl_length n);</pre>	<p>Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported:</p> <p><code>csipl_ccorrelate1d_split_f</code> <code>csipl_ccorrelate1d_split_d</code></p>
<pre>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);</pre>	<p>Create a 2D correlation object. The following instances are supported:</p> <p><code>csipl_corr2d_create_f</code> <code>csipl_corr2d_create_d</code> <code>csipl_ccorr2d_create_f</code> <code>csipl_ccorr2d_create_d</code></p>
<pre>int csipl_Dcorr2d_destroy_P(csipl_Dcorr2d_P * plan);</pre>	<p>Destroy a 2D correlation object. The following instances are supported:</p> <p><code>csipl_corr2d_destroy_f</code> <code>csipl_corr2d_destroy_d</code> <code>csipl_ccorr2d_destroy_f</code> <code>csipl_ccorr2d_destroy_d</code></p>
<pre>void csipl_Dcorr2d_getattr_P(csipl_Dcorr2d_P * plan, csipl_Dcorr2d_attr_P * attr);</pre>	<p>Return the attributes for a 2D correlation object. The following instances are supported:</p> <p><code>csipl_corr2d_getattr_f</code> <code>csipl_corr2d_getattr_d</code> <code>csipl_ccorr2d_getattr_f</code> <code>csipl_ccorr2d_getattr_d</code></p>

Prototype	Description
<pre>void csipl_correlate2d_P(csipl_corr2d_P * plan, csipl_bias bias, scalar_P * ref, int ldref, scalar_P * x, int ldx, scalar_P * y, int ldy, csipl_length m, csipl_length n);</pre>	<p>Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported:</p> <p><code>csipl_correlate2d_f</code> <code>csipl_correlate2d_d</code></p>
<pre>void csipl_ccorrelate2d_inter_P(csipl_ccorr2d_P * plan, csipl_bias bias, void * ref, int ldref, void * x, int ldx, void * y, int ldy, csipl_length m, csipl_length n);</pre>	<p>Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported:</p> <p><code>csipl_ccorrelate2d_inter_f</code> <code>csipl_ccorrelate2d_inter_d</code></p>
<pre>void csipl_ccorrelate2d_split_P(csipl_ccorr2d_P * plan, csipl_bias bias, scalar_P * ref_re, scalar_P * ref_im, int ldref, scalar_P * x_re, scalar_P * x_im, int ldx, scalar_P * y_re, scalar_P * y_im, int ldy, csipl_length m, csipl_length n);</pre>	<p>Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported:</p> <p><code>csipl_ccorrelate2d_split_f</code> <code>csipl_ccorrelate2d_split_d</code></p>

7.3 Window Functions

Prototype	Description
<pre>scalar_P * csipl_vcreate_blackman_P(unsigned int N, csipl_memory_hint hint);</pre>	<p>Create a vector with Blackman window weights. The following instances are supported:</p> <p><code>csipl_vcreate_blackman_f</code> <code>csipl_vcreate_blackman_d</code></p>
<pre>scalar_P * csipl_vcreate_cheby_P(unsigned int N, scalar_P ripple, csipl_memory_hint hint);</pre>	<p>Create a vector with Dolph-Chebyshev window weights. The following instances are supported:</p> <p><code>csipl_vcreate_cheby_f</code> <code>csipl_vcreate_cheby_d</code></p>

Prototype	Description
<pre>scalar_P * csipl_vcreate_hanning_P(unsigned int N, csipl_memory_hint hint);</pre>	<p>Create a vector with Hanning window weights. The following instances are supported:</p> <pre>csipl_vcreate_hanning_f csipl_vcreate_hanning_d</pre>
<pre>scalar_P * csipl_vcreate_kaiser_P(unsigned int N, scalar_P beta, csipl_memory_hint hint);</pre>	<p>Create a vector with Kaiser window weights. The following instances are supported:</p> <pre>csipl_vcreate_kaiser_f csipl_vcreate_kaiser_d</pre>

7.4 Filter Functions

Prototype	Description
<pre>csipl_fir_P * csipl_fir_create_P(scalar_P * 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);</pre>	<p>Create a decimated FIR filter object. The following instances are supported:</p> <pre>csipl_fir_create_f csipl_fir_create_d</pre>
<pre>csipl_cfir_P * csipl_cfir_create_inter_P(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);</pre>	<p>Create a decimated FIR filter object. The following instances are supported:</p> <pre>csipl_cfir_create_inter_f csipl_cfir_create_inter_d</pre>
<pre>csipl_cfir_P * csipl_cfir_create_split_P(scalar_P * kernel_re, scalar_P * 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);</pre>	<p>Create a decimated FIR filter object. The following instances are supported:</p> <pre>csipl_cfir_create_split_f csipl_cfir_create_split_d</pre>
<pre>int csipl_Dfir_destroy_P(csipl_Dfir_P * plan);</pre>	<p>Destroy a FIR filter object. The following instances are supported:</p> <pre>csipl_fir_destroy_f csipl_fir_destroy_d csipl_cfir_destroy_f csipl_cfir_destroy_d</pre>

Prototype	Description
<pre>int csipl_firflt_P(csipl_fir_P * plan, scalar_P * x, csipl_stride stridex, scalar_P * y, csipl_stride stridey, csipl_length n);</pre>	<p>FIR filter an input sequence and decimate the output. The following instances are supported:</p> <p><code>csipl_firflt_f</code> <code>csipl_firflt_d</code></p>
<pre>int csipl_cfirflt_inter_P(csipl_cfir_P * plan, void * x, csipl_stride stridex, void * y, csipl_stride stridey, csipl_length n);</pre>	<p>FIR filter an input sequence and decimate the output. The following instances are supported:</p> <p><code>csipl_cfirflt_inter_f</code> <code>csipl_cfirflt_inter_d</code></p>
<pre>int csipl_cfirflt_split_P(csipl_cfir_P * plan, scalar_P * x_re, scalar_P * x_im, csipl_stride stridex, scalar_P * y_re, scalar_P * y_im, csipl_stride stridey, csipl_length n);</pre>	<p>FIR filter an input sequence and decimate the output. The following instances are supported:</p> <p><code>csipl_cfirflt_split_f</code> <code>csipl_cfirflt_split_d</code></p>
<pre>void csipl_Dfir_getattr_P(csipl_Dfir_P * plan, csipl_Dfir_attr_P * attr);</pre>	<p>Return the attributes of a FIR filter object. The following instances are supported:</p> <p><code>csipl_fir_getattr_f</code> <code>csipl_fir_getattr_d</code> <code>csipl_cfir_getattr_f</code> <code>csipl_cfir_getattr_d</code></p>
<pre>void csipl_Dfir_reset_P(csipl_Dfir_P * fir);</pre>	<p>Reset the state of a decimated FIR filter object. The following instances are supported:</p> <p><code>csipl_fir_reset_f</code> <code>csipl_fir_reset_d</code> <code>csipl_cfir_reset_f</code> <code>csipl_cfir_reset_d</code></p>

7.5 Miscellaneous Signal Processing Functions

Prototype	Description
<pre>void csipl_vhisto_P(scalar_P * A, csipl_stride strideA, scalar_P min, scalar_P max, csipl_hist_opt opt, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the histogram of a vector. The following instances are supported:</p> <pre>csipl_vhisto_f csipl_vhisto_d</pre>

Chapter 8. Linear Algebra

8.1 Matrix and Vector Operations

Prototype	Description
<pre>void csipl_cmherm_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Complex Hermitian (conjugate transpose) of a matrix. The following instances are supported: <code>csipl_cmherm_inter_f</code> <code>csipl_cmherm_inter_d</code>
<pre>void csipl_cmherm_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);</pre>	Complex Hermitian (conjugate transpose) of a matrix. The following instances are supported: <code>csipl_cmherm_split_f</code> <code>csipl_cmherm_split_d</code>
<pre>csipl_cscalar_P csipl_cvjdot_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_length n);</pre>	Compute the conjugate inner (dot) product of two complex vectors. The following instances are supported: <code>csipl_cvjdot_inter_f</code> <code>csipl_cvjdot_inter_d</code>
<pre>csipl_cscalar_P csipl_cvjdot_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, csipl_length n);</pre>	Compute the conjugate inner (dot) product of two complex vectors. The following instances are supported: <code>csipl_cvjdot_split_f</code> <code>csipl_cvjdot_split_d</code>
<pre>void csipl_gemp_P(scalar_P alpha, scalar_P * A, int ldA, csipl_mat_op Aop, scalar_P * B, int ldB, csipl_mat_op Bop, scalar_P beta, scalar_P * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	Calculate the general product of two matrices and accumulate. The following instances are supported: <code>csipl_gemp_f</code> <code>csipl_gemp_d</code>

Prototype	Description
<pre>void csipl_cgemp_inter_P(csipl_cscalar_P alpha, void * A, int ldA, csipl_mat_op Aop, void * B, int ldB, csipl_mat_op Bop, csipl_cscalar_P beta, void * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the general product of two matrices and accumulate. The following instances are supported:</p> <pre>csipl_cgemp_inter_f csipl_cgemp_inter_d</pre>
<pre>void csipl_cgemp_split_P(csipl_cscalar_P alpha_re, csipl_cscalar_P alpha_im, scalar_P * A_re, scalar_P * A_im, int ldA, csipl_mat_op Aop, scalar_P * B_re, scalar_P * B_im, int ldB, csipl_mat_op Bop, csipl_cscalar_P beta_re, csipl_cscalar_P beta_im, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the general product of two matrices and accumulate. The following instances are supported:</p> <pre>csipl_cgemp_split_f csipl_cgemp_split_d</pre>
<pre>void csipl_gems_P(scalar_P alpha, scalar_P * A, int ldA, csipl_mat_op Aop, scalar_P beta, scalar_P * C, int ldC, csipl_length m, csipl_length n);</pre>	<p>Calculate a general matrix sum. The following instances are supported:</p> <pre>csipl_gems_f csipl_gems_d</pre>
<pre>void csipl_cgems_inter_P(csipl_cscalar_P alpha, void * A, int ldA, csipl_mat_op Aop, csipl_cscalar_P beta, void * C, int ldC, csipl_length m, csipl_length n);</pre>	<p>Calculate a general matrix sum. The following instances are supported:</p> <pre>csipl_cgems_inter_f csipl_cgems_inter_d</pre>

Prototype	Description
<pre>void csipl_cgems_split_P(csipl_cscalar_P alpha_re, csipl_cscalar_P alpha_im, scalar_P * A_re, scalar_P * A_im, int ldA, csipl_mat_op Aop, csipl_cscalar_P beta_re, csipl_cscalar_P beta_im, scalar_P * C_re, scalar_P * C_im, int ldC, csipl_length m, csipl_length n);</pre>	<p>Calculate a general matrix sum. The following instances are supported:</p> <pre>csipl_cgems_split_f csipl_cgems_split_d</pre>
<pre>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);</pre>	<p>Calculate the product of two matrices. The following instances are supported:</p> <pre>csipl_mprod_f csipl_mprod_d csipl_mprod_i csipl_mprod_si</pre>
<pre>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);</pre>	<p>Calculate the product of two matrices. The following instances are supported:</p> <pre>csipl_cmprod_inter_f csipl_cmprod_inter_d csipl_cmprod_inter_i csipl_cmprod_inter_si</pre>
<pre>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);</pre>	<p>Calculate the product of two matrices. The following instances are supported:</p> <pre>csipl_cmprod_split_f csipl_cmprod_split_d csipl_cmprod_split_i csipl_cmprod_split_si</pre>
<pre>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);</pre>	<p>Calculate the product a complex matrix and the Hermitian of a complex matrix. The following instances are supported:</p> <pre>csipl_cmprodh_inter_f csipl_cmprodh_inter_d csipl_cmprodh_inter_i csipl_cmprodh_inter_si</pre>

Prototype	Description
<pre>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);</pre>	<p>Calculate the product a complex matrix and the Hermitian of a complex matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodh_split_f csipl_cmprodh_split_d csipl_cmprodh_split_i csipl_cmprodh_split_si</pre>
<pre>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);</pre>	<p>Calculate the product a complex matrix and the conjugate of a complex matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodj_inter_f csipl_cmprodj_inter_d csipl_cmprodj_inter_i csipl_cmprodj_inter_si</pre>
<pre>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);</pre>	<p>Calculate the product a complex matrix and the conjugate of a complex matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodj_split_f csipl_cmprodj_split_d csipl_cmprodj_split_i csipl_cmprodj_split_si</pre>
<pre>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);</pre>	<p>Calculate the product of a matrix and the transpose of a matrix.</p> <p>The following instances are supported:</p> <pre>csipl_mprodt_f csipl_mprodt_d csipl_mprodt_i csipl_mprodt_si</pre>
<pre>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);</pre>	<p>Calculate the product of a matrix and the transpose of a matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodt_inter_f csipl_cmprodt_inter_d csipl_cmprodt_inter_i csipl_cmprodt_inter_si</pre>

Prototype	Description
<pre>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);</pre>	<p>Calculate the product of a matrix and the transpose of a matrix. The following instances are supported:</p> <pre>csipl_cmprodt_split_f csipl_cmprodt_split_d csipl_cmprodt_split_i csipl_cmprodt_split_si</pre>
<pre>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);</pre>	<p>Calculate a matrix–vector product. The following instances are supported:</p> <pre>csipl_mvprod_f csipl_mvprod_d csipl_mvprod_i csipl_mvprod_si</pre>
<pre>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);</pre>	<p>Calculate a matrix–vector product. The following instances are supported:</p> <pre>csipl_cmvprod_inter_f csipl_cmvprod_inter_d csipl_cmvprod_inter_i csipl_cmvprod_inter_si</pre>
<pre>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);</pre>	<p>Calculate a matrix–vector product. The following instances are supported:</p> <pre>csipl_cmvprod_split_f csipl_cmvprod_split_d csipl_cmvprod_split_i csipl_cmvprod_split_si</pre>
<pre>void csipl_mtrans_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Transpose a matrix. The following instances are supported:</p> <pre>csipl_mtrans_bl csipl_mtrans_f csipl_mtrans_d csipl_mtrans_i csipl_mtrans_si</pre>

Prototype	Description
<pre>void csipl_cmtrans_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Transpose a matrix. The following instances are supported:</p> <pre>csipl_cmtrans_inter_f csipl_cmtrans_inter_d csipl_cmtrans_inter_i csipl_cmtrans_inter_si</pre>
<pre>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);</pre>	<p>Transpose a matrix. The following instances are supported:</p> <pre>csipl_cmtrans_split_f csipl_cmtrans_split_d csipl_cmtrans_split_i csipl_cmtrans_split_si</pre>
<pre>scalar_P csipl_vdot_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, csipl_length n);</pre>	<p>Compute the inner (dot) product of two vectors. The following instances are supported:</p> <pre>csipl_vdot_f csipl_vdot_d</pre>
<pre>csipl_cscalar_P csipl_cvdot_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_length n);</pre>	<p>Compute the inner (dot) product of two vectors. The following instances are supported:</p> <pre>csipl_cvdot_inter_f csipl_cvdot_inter_d</pre>
<pre>csipl_cscalar_P csipl_cvdot_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, csipl_length n);</pre>	<p>Compute the inner (dot) product of two vectors. The following instances are supported:</p> <pre>csipl_cvdot_split_f csipl_cvdot_split_d</pre>
<pre>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);</pre>	<p>Calculate a vector–matrix product. The following instances are supported:</p> <pre>csipl_vmprod_f csipl_vmprod_d csipl_vmprod_i csipl_vmprod_si</pre>
<pre>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);</pre>	<p>Calculate a vector–matrix product. The following instances are supported:</p> <pre>csipl_cvmprod_inter_f csipl_cvmprod_inter_d csipl_cvmprod_inter_i csipl_cvmprod_inter_si</pre>

Prototype	Description
<pre>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);</pre>	<p>Calculate a vector–matrix product. The following instances are supported:</p> <p><code>csipl_cvmprod_split_f</code> <code>csipl_cvmprod_split_d</code> <code>csipl_cvmprod_split_i</code> <code>csipl_cvmprod_split_si</code></p>
<pre>void csipl_vouter_P(scalar_P alpha, scalar_P * X, csipl_stride strideX, scalar_P * Y, csipl_stride strideY, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Calculate the outer product of two vectors. The following instances are supported:</p> <p><code>csipl_vouter_f</code> <code>csipl_vouter_d</code></p>
<pre>void csipl_cvouter_inter_P(csipl_cscalar_P alpha, void * X, csipl_stride strideX, void * Y, csipl_stride strideY, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Calculate the outer product of two vectors. The following instances are supported:</p> <p><code>csipl_cvouter_inter_f</code> <code>csipl_cvouter_inter_d</code></p>
<pre>void csipl_cvouter_split_P(csipl_cscalar_P alpha_re, csipl_cscalar_P alpha_im, scalar_P * X_re, scalar_P * X_im, csipl_stride strideX, scalar_P * Y_re, scalar_P * Y_im, csipl_stride strideY, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Calculate the outer product of two vectors. The following instances are supported:</p> <p><code>csipl_cvouter_split_f</code> <code>csipl_cvouter_split_d</code></p>
<pre>scalar_P csipl_vcsummgval_inter_P(void * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the magnitudes of the elements of a complex vector. The following instances are supported:</p> <p><code>csipl_vcsummgval_inter_f</code> <code>csipl_vcsummgval_inter_d</code></p>
<pre>scalar_P csipl_vcsummgval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the magnitudes of the elements of a complex vector. The following instances are supported:</p> <p><code>csipl_vcsummgval_split_f</code> <code>csipl_vcsummgval_split_d</code></p>

Prototype	Description
<pre>void csipl_minvlu_P(scalar_P * A, int ldA, signed int * V, csipl_stride strideV, scalar_P * R, int ldR, csipl_length n);</pre>	<p>Invert a square matrix using LU decomposition.</p> <p>The following instances are supported:</p> <pre>csipl_minvlu_f csipl_minvlu_d</pre>
<pre>void csipl_cminvlu_inter_P(void * A, int ldA, signed int * V, csipl_stride strideV, void * R, int ldR, csipl_length n);</pre>	<p>Invert a square matrix using LU decomposition.</p> <p>The following instances are supported:</p> <pre>csipl_cminvlu_inter_f csipl_cminvlu_inter_d</pre>
<pre>void csipl_cminvlu_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, signed int * V, csipl_stride strideV, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length n);</pre>	<p>Invert a square matrix using LU decomposition.</p> <p>The following instances are supported:</p> <pre>csipl_cminvlu_split_f csipl_cminvlu_split_d</pre>

8.2 Special Linear System Solvers

Prototype	Description
<pre>int csipl_covsol_P(scalar_P * A, int ldA, scalar_P * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Solve a covariance linear system problem.</p> <p>The following instances are supported:</p> <p><code>csipl_covsol_f</code> <code>csipl_covsol_d</code></p>
<pre>int csipl_ccovsol_inter_P(void * A, int ldA, void * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Solve a covariance linear system problem.</p> <p>The following instances are supported:</p> <p><code>csipl_ccovsol_inter_f</code> <code>csipl_ccovsol_inter_d</code></p>
<pre>int csipl_ccovsol_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * XB_re, scalar_P * XB_im, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Solve a covariance linear system problem.</p> <p>The following instances are supported:</p> <p><code>csipl_ccovsol_split_f</code> <code>csipl_ccovsol_split_d</code></p>
<pre>int csipl_llsqsol_P(scalar_P * A, int ldA, scalar_P * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Solve a linear least squares problem.</p> <p>The following instances are supported:</p> <p><code>csipl_llsqsol_f</code> <code>csipl_llsqsol_d</code></p>
<pre>int csipl_cllsqsol_inter_P(void * A, int ldA, void * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Solve a linear least squares problem.</p> <p>The following instances are supported:</p> <p><code>csipl_cllsqsol_inter_f</code> <code>csipl_cllsqsol_inter_d</code></p>
<pre>int csipl_cllsqsol_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * XB_re, scalar_P * XB_im, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Solve a linear least squares problem.</p> <p>The following instances are supported:</p> <p><code>csipl_cllsqsol_split_f</code> <code>csipl_cllsqsol_split_d</code></p>

Prototype	Description
<pre>int csipl_toepsol_P(scalar_P * T, csipl_stride strideT, scalar_P * B, csipl_stride strideB, scalar_P * W, csipl_stride strideW, scalar_P * X, csipl_stride strideX, csipl_length n);</pre>	<p>Solve a real symmetric positive definite Toeplitz linear system. The following instances are supported:</p> <p><code>csipl_toepsol_f</code> <code>csipl_toepsol_d</code></p>
<pre>int csipl_ctoepsol_inter_P(void * T, csipl_stride strideT, void * B, csipl_stride strideB, void * W, csipl_stride strideW, void * X, csipl_stride strideX, csipl_length n);</pre>	<p>Solve a real symmetric positive definite Toeplitz linear system. The following instances are supported:</p> <p><code>csipl_ctoepsol_inter_f</code> <code>csipl_ctoepsol_inter_d</code></p>
<pre>int csipl_ctoepsol_split_P(scalar_P * T_re, scalar_P * T_im, csipl_stride strideT, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, scalar_P * W_re, scalar_P * W_im, csipl_stride strideW, scalar_P * X_re, scalar_P * X_im, csipl_stride strideX, csipl_length n);</pre>	<p>Solve a real symmetric positive definite Toeplitz linear system. The following instances are supported:</p> <p><code>csipl_ctoepsol_split_f</code> <code>csipl_ctoepsol_split_d</code></p>

8.3 General Square Linear System Solver

Prototype	Description
<pre>int csipl_lud_P(csipl_clu_P * lud, scalar_P * A, int ldA, csipl_length n);</pre>	<p>Compute an LU decomposition of a square matrix using partial pivoting. The following instances are supported:</p> <p><code>csipl_lud_f</code> <code>csipl_lud_d</code></p>
<pre>int csipl_clud_inter_P(csipl_clu_P * lud, void * A, int ldA, csipl_length n);</pre>	<p>Compute an LU decomposition of a square matrix using partial pivoting. The following instances are supported:</p> <p><code>csipl_clud_inter_f</code> <code>csipl_clud_inter_d</code></p>
<pre>int csipl_clud_split_P(csipl_clu_P * lud, scalar_P * A_re, scalar_P * A_im, int ldA, csipl_length n);</pre>	<p>Compute an LU decomposition of a square matrix using partial pivoting. The following instances are supported:</p> <p><code>csipl_clud_split_f</code> <code>csipl_clud_split_d</code></p>

Prototype	Description
<pre> <i>csipl_Dlu_P</i> * csipl_Dlud_create_P(unsigned int <i>N</i>); </pre>	<p>Create an LU decomposition object. The following instances are supported:</p> <pre> csipl_lud_create_f csipl_lud_create_d csipl_clud_create_f csipl_clud_create_d </pre>
<pre> int csipl_Dlud_destroy_P(<i>csipl_Dlu_P</i> * <i>lud</i>); </pre>	<p>Destroy an LU decomposition object. The following instances are supported:</p> <pre> csipl_lud_destroy_f csipl_lud_destroy_d csipl_clud_destroy_f csipl_clud_destroy_d </pre>
<pre> void csipl_Dlud_getattr_P(<i>csipl_Dlu_P</i> * <i>lud</i>, <i>csipl_Dlu_attr_P</i> * <i>attr</i>); </pre>	<p>Returns the attributes of an LU decomposition object. The following instances are supported:</p> <pre> csipl_lud_getattr_f csipl_lud_getattr_d csipl_clud_getattr_f csipl_clud_getattr_d </pre>
<pre> int csipl_lusol_P(<i>csipl_clu_P</i> * <i>clud</i>, <i>csipl_mat_op</i> <i>opA</i>, <i>scalar_P</i> * <i>XB</i>, int <i>ldXB</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	<p>Solve a square linear system. The following instances are supported:</p> <pre> csipl_lusol_f csipl_lusol_d </pre>
<pre> int csipl_clusol_inter_P(<i>csipl_clu_P</i> * <i>clud</i>, <i>csipl_mat_op</i> <i>opA</i>, void * <i>XB</i>, int <i>ldXB</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	<p>Solve a square linear system. The following instances are supported:</p> <pre> csipl_clusol_inter_f csipl_clusol_inter_d </pre>
<pre> int csipl_clusol_split_P(<i>csipl_clu_P</i> * <i>clud_re</i>, <i>csipl_clu_P</i> * <i>clud_im</i>, <i>csipl_mat_op</i> <i>opA</i>, <i>scalar_P</i> * <i>XB_re</i>, <i>scalar_P</i> * <i>XB_im</i>, int <i>ldXB</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	<p>Solve a square linear system. The following instances are supported:</p> <pre> csipl_clusol_split_f csipl_clusol_split_d </pre>

8.4 Symmetric Positive Definite Linear System Solver

Prototype	Description
<pre> int csipl_chold_P(<i>csipl_cchol_P</i> * <i>chold</i>, <i>scalar_P</i> * <i>A</i>, int <i>ldA</i>, <i>csipl_length</i> <i>n</i>); </pre>	<p>Compute a Cholesky decomposition of a symmetric positive definite matrix. The following instances are supported:</p> <pre> csipl_chold_f csipl_chold_d </pre>

Prototype	Description
<pre>int csipl_cchold_inter_P(csipl_cchol_P * chold, void * A, int ldA, csipl_length n);</pre>	<p>Compute a Cholesky decomposition of a symmetric positive definite matrix. The following instances are supported:</p> <pre>csipl_cchold_inter_f csipl_cchold_inter_d</pre>
<pre>int csipl_cchold_split_P(csipl_cchol_P * chold, scalar_P * A_re, scalar_P * A_im, int ldA, csipl_length n);</pre>	<p>Compute a Cholesky decomposition of a symmetric positive definite matrix. The following instances are supported:</p> <pre>csipl_cchold_split_f csipl_cchold_split_d</pre>
<pre>csipl_chol_P * csipl_chold_create_P(csipl_mat_uplo uplo, unsigned int n);</pre>	<p>Creates a Cholesky decomposition object. The following instances are supported:</p> <pre>csipl_chold_create_f csipl_chold_create_d</pre>
<pre>csipl_cchol_P * csipl_cchold_create_P(csipl_mat_uplo uplo, unsigned int n);</pre>	<p>Creates a Cholesky decomposition object. The following instances are supported:</p> <pre>csipl_cchold_create_f csipl_cchold_create_d</pre>
<pre>int csipl_Dchold_destroy_P(csipl_Dchol_P * chold);</pre>	<p>Destroy a Cholesky decomposition object. The following instances are supported:</p> <pre>csipl_chold_destroy_f csipl_chold_destroy_d csipl_cchold_destroy_f csipl_cchold_destroy_d</pre>
<pre>void csipl_Dchold_getattr_P(csipl_Dchol_P * chold, csipl_Dchol_attr_P * attr);</pre>	<p>Returns the attributes of a Cholesky decomposition object. The following instances are supported:</p> <pre>csipl_chold_getattr_f csipl_chold_getattr_d csipl_cchold_getattr_f csipl_cchold_getattr_d</pre>
<pre>int csipl_cholsol_P(csipl_cchol_P * chold, scalar_P * XB, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve a symmetric positive definite linear system. The following instances are supported:</p> <pre>csipl_cholsol_f csipl_cholsol_d</pre>
<pre>int csipl_ccholsol_inter_P(csipl_cchol_P * chold, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve a symmetric positive definite linear system. The following instances are supported:</p> <pre>csipl_ccholsol_inter_f csipl_ccholsol_inter_d</pre>

Prototype	Description
<pre>int csipl_ccholsol_split_P(csipl_cchol_P * chold, scalar_P * XB_re, scalar_P * XB_im, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve a symmetric positive definite linear system.</p> <p>The following instances are supported:</p> <pre>csipl_ccholsol_split_f csipl_ccholsol_split_d</pre>

8.5 Overdetermined Linear System Solver

Prototype	Description
<pre>int csipl_qrd_P(csipl_cqr_P * qrd, scalar_P * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Compute a QR decomposition of a matrix .</p> <p>The following instances are supported:</p> <pre>csipl_qrd_f csipl_qrd_d</pre>
<pre>int csipl_cqrd_inter_P(csipl_cqr_P * qrd, void * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Compute a QR decomposition of a matrix .</p> <p>The following instances are supported:</p> <pre>csipl_cqrd_inter_f csipl_cqrd_inter_d</pre>
<pre>int csipl_cqrd_split_P(csipl_cqr_P * qrd, scalar_P * A_re, scalar_P * A_im, int ldA, csipl_length m, csipl_length n);</pre>	<p>Compute a QR decomposition of a matrix .</p> <p>The following instances are supported:</p> <pre>csipl_cqrd_split_f csipl_cqrd_split_d</pre>
<pre>csipl_qr_P * csipl_qrd_create_P(unsigned int m, unsigned int n, csipl_qrd_qopt qopt);</pre>	<p>Create a QR decomposition object.</p> <p>The following instances are supported:</p> <pre>csipl_qrd_create_f csipl_qrd_create_d</pre>
<pre>csipl_cqr_P * csipl_cqrd_create_P(unsigned int m, unsigned int n, csipl_qrd_qopt qopt);</pre>	<p>Create a QR decomposition object.</p> <p>The following instances are supported:</p> <pre>csipl_cqrd_create_f csipl_cqrd_create_d</pre>
<pre>int csipl_Dqrd_destroy_P(csipl_Dqr_P * qrd);</pre>	<p>Destroy a QR decomposition object.</p> <p>The following instances are supported:</p> <pre>csipl_qrd_destroy_f csipl_qrd_destroy_d csipl_cqrd_destroy_f csipl_cqrd_destroy_d</pre>
<pre>void csipl_Dqrd_getattr_P(csipl_Dqr_P * qrd, csipl_Dqr_attr_P * attr);</pre>	<p>Returns the attributes of a QR decomposition object.</p> <p>The following instances are supported:</p> <pre>csipl_qrd_getattr_f csipl_qrd_getattr_d csipl_cqrd_getattr_f csipl_cqrd_getattr_d</pre>

Prototype	Description
<pre>int csipl_qrdprodq_P(csipl_qr_P * qrd, csipl_mat_op opQ, csipl_mat_side apQ, scalar_P * C, int ldC, csipl_length m, csipl_length n);</pre>	<p>Multiply a matrix by the matrix Q from a QR decomposition. The following instances are supported:</p> <p><code>csipl_qrdprodq_f</code> <code>csipl_qrdprodq_d</code></p>
<pre>int csipl_cqrdprodq_inter_P(csipl_cqr_P * qrd, csipl_mat_op opQ, csipl_mat_side apQ, void * C, int ldC, csipl_length m, csipl_length n);</pre>	<p>Multiply a matrix by the matrix Q from a QR decomposition. The following instances are supported:</p> <p><code>csipl_cqrdprodq_inter_f</code> <code>csipl_cqrdprodq_inter_d</code></p>
<pre>int csipl_cqrdprodq_split_P(csipl_cqr_P * qrd_re, csipl_cqr_P * qrd_im, csipl_mat_op opQ, csipl_mat_side apQ, scalar_P * C_re, scalar_P * C_im, int ldC, csipl_length m, csipl_length n);</pre>	<p>Multiply a matrix by the matrix Q from a QR decomposition. The following instances are supported:</p> <p><code>csipl_cqrdprodq_split_f</code> <code>csipl_cqrdprodq_split_d</code></p>
<pre>int csipl_qrdsolr_P(csipl_qr_P * qrd, csipl_mat_op OpR, scalar_P alpha, scalar_P * XB, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve linear system based on the matrix R, from QR decomposition of the matrix A. The following instances are supported:</p> <p><code>csipl_qrdsolr_f</code> <code>csipl_qrdsolr_d</code></p>
<pre>int csipl_cqrdsolr_inter_P(csipl_cqr_P * qrd, csipl_mat_op OpR, csipl_cscalar_P alpha, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve linear system based on the matrix R, from QR decomposition of the matrix A. The following instances are supported:</p> <p><code>csipl_cqrdsolr_inter_f</code> <code>csipl_cqrdsolr_inter_d</code></p>
<pre>int csipl_cqrdsolr_split_P(csipl_cqr_P * qrd_re, csipl_cqr_P * qrd_im, csipl_mat_op OpR, csipl_cscalar_P alpha_re, csipl_cscalar_P alpha_im, scalar_P * XB_re, scalar_P * XB_im, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve linear system based on the matrix R, from QR decomposition of the matrix A. The following instances are supported:</p> <p><code>csipl_cqrdsolr_split_f</code> <code>csipl_cqrdsolr_split_d</code></p>

Prototype	Description
<pre>int csipl_qrsol_P(csipl_qr_P * qrd, csipl_mat_op OpR, scalar_P alpha, scalar_P * XB, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve linear system based on the matrix R, from QR decomposition of the matrix A.</p> <p>The following instances are supported:</p> <p><code>csipl_qrsol_f</code> <code>csipl_qrsol_d</code></p>
<pre>int csipl_cqrsol_inter_P(csipl_cqr_P * qrd, csipl_qrd_prob prob, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve either a linear covariance or linear least squares problem.</p> <p>The following instances are supported:</p> <p><code>csipl_cqrsol_inter_f</code> <code>csipl_cqrsol_inter_d</code></p>
<pre>int csipl_cqrsol_split_P(csipl_cqr_P * qrd_re, csipl_cqr_P * qrd_im, csipl_qrd_prob prob, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	<p>Solve either a linear covariance or linear least squares problem.</p> <p>The following instances are supported:</p> <p><code>csipl_cqrsol_split_f</code> <code>csipl_cqrsol_split_d</code></p>