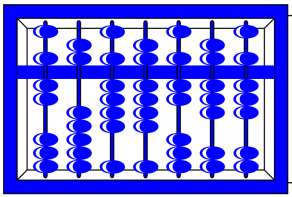

NASoftware Limited

Incorporating InfoSAR

CSIPL
List of Contents

CSIPL/Contents [3.1D]

Release 3.1D
May 2013

Chapter 1. Support Functions

1.1 Initilialisation and Finalisation

Prototype	Description
`int csipl_init( void * ptr);`	Provides initialisation, allowing the library to allocate and set a global state, and prepare to support the use of CSIPL functionality by the user.
`int csipl_finalize( void * ptr);`	Provides cleanup and releases resources used by CSIPL (if the last of a nested series of calls), allowing the library to guarantee that any resources allocated by `csipl_init` are no longer in use after the call is complete.

1.2 Sundry Functions

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

Chapter 2. Scalar Functions

2.1 Real Scalar Functions

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

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

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

2.2 Complex Scalar Functions

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

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

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

2.3 Index Scalar Functions

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

Prototype	Description
`csipl_index csipl_mrowindex( csipl_scalar_mi mi);`	Returns the row vector index from a matrix index.

Chapter 3. Random Number Generation

3.1 Random Number Functions

Prototype	Description
csipl_randstate * csipl_randcreate(csipl_index seed, csipl_index numprocs, csipl_index id, csipl_rng portable);	Create a random number generator state object.
int csipl_randdestroy(csipl_randstate * rand);	Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure.
scalar_P csipl_randu_P(csipl_randstate * state);	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: csipl_randu_f csipl_randu_d
csipl_cscalar_P csipl_crandu_P(csipl_randstate * state);	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: csipl_crandu_f csipl_crandu_d
void csipl_vrandu_P(csipl_randstate * state, scalar_P * R, csipl_stride strideR, csipl_length n);	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: csipl_vrandu_f csipl_vrandu_d

Prototype	Description
void csipl_cvrandu_inter_P(csipl_randstate * state, void * R, csipl_stride strideR, csipl_length n);	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: csipl_cvrandu_inter_f csipl_cvrandu_inter_d
void csipl_cvrandu_split_P(csipl_randstate * state, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);	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: csipl_cvrandu_split_f csipl_cvrandu_split_d
scalar_P csipl_randn_P(csipl_randstate * state);	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: csipl_randn_f csipl_randn_d
csipl_cscalar_P csipl_crandn_P(csipl_randstate * state);	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: csipl_crandn_f csipl_crandn_d
scalar_P csipl_vrandn_P(csipl_randstate * state);	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: csipl_vrandn_f csipl_vrandn_d

Prototype	Description
void csipl_cvrandn_inter_P(csipl_randstate * state, void * R, csipl_stride strideR, csipl_length n);	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: csipl_cvrandn_inter_f csipl_cvrandn_inter_d
void csipl_cvrandn_split_P(csipl_randstate * state, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);	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: csipl_cvrandn_split_f csipl_cvrandn_split_d

Chapter 4. Vector And Elementwise Operations

4.1 Elementary Mathematical Functions

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

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

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

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

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

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

4.2 Unary Operations

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

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

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

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

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

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

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

Prototype	Description
void csipl_mrsqrt_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);	Computes the reciprocal of the square root for each element of a matrix. The following instances are supported: csipl_mrsqrt_f csipl_mrsqrt_d
void csipl_vsq_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);	Computes the square for each element of a vector. The following instances are supported: csipl_vsq_f csipl_vsq_d
void csipl_msq_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);	Computes the square for each element of a matrix. The following instances are supported: csipl_msq_f csipl_msq_d
scalar_P csipl_cvsumval_inter_P(scalar_P * A, csipl_stride strideA, csipl_length n);	Returns the sum of the elements of a vector. The following instances are supported: csipl_cvsumval_inter_f csipl_cvsumval_inter_d csipl_cvsumval_inter_i
scalar_P csipl_cvsumval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);	Returns the sum of the elements of a vector. The following instances are supported: csipl_cvsumval_split_f csipl_cvsumval_split_d csipl_cvsumval_split_i
scalar_P csipl_msumval_P(scalar_P * A, csipl_stride strideA, csipl_length n);	Returns the sum of the elements of a vector. The following instances are supported: csipl_msumval_f csipl_msumval_d csipl_msumval_i
scalar_P csipl_cmsumval_inter_P(scalar_P * A, csipl_stride strideA, csipl_length n);	Returns the sum of the elements of a vector. The following instances are supported: csipl_cmsumval_inter_f csipl_cmsumval_inter_d csipl_cmsumval_inter_i
scalar_P csipl_cmsumval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);	Returns the sum of the elements of a vector. The following instances are supported: csipl_cmsumval_split_f csipl_cmsumval_split_d csipl_cmsumval_split_i

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

4.3 Binary Operations

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

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

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

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

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

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

Prototype	Description
void csipl_vsdiv_P(scalar_P * A, csipl_stride strideA, scalar_P b, scalar_P * R, csipl_stride strideR, csipl_length n);	Computes the quotient, by element, of a vector and a scalar. The following instances are supported: csipl_vsdiv_f csipl_vsdiv_d
void csipl_cvrsdiv_inter_P(scalar_P * A, csipl_stride strideA, scalar_P b, scalar_P * R, csipl_stride strideR, csipl_length n);	Computes the quotient, by element, of a vector and a scalar. The following instances are supported: csipl_cvrsdiv_inter_f csipl_cvrsdiv_inter_d
void csipl_cvrsdiv_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);	Computes the quotient, by element, of a vector and a scalar. The following instances are supported: csipl_cvrsdiv_split_f csipl_cvrsdiv_split_d
void csipl_msdiv_P(scalar_P * A, int ldA, scalar_P b, scalar_P * R, int ldR, csipl_length m, csipl_length n);	Computes the quotient, by element, of a matrix and a scalar. The following instances are supported: csipl_msdiv_f csipl_msdiv_d
void csipl_cmrsdiv_inter_P(void * A, int ldA, csipl_cscalar_P b, void * R, int ldR, csipl_length m, csipl_length n);	Computes the quotient, by element, of a matrix and a scalar. The following instances are supported: csipl_cmrsdiv_inter_f csipl_cmrsdiv_inter_d
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);	Computes the quotient, by element, of a matrix and a scalar. The following instances are supported: csipl_cmrsdiv_split_f csipl_cmrsdiv_split_d

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

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

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

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

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

Prototype	Description
void csipl_rscmmul_inter_P(scalar_P a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);	Computes the product, by element, of a real scalar and a complex matrix. The following instances are supported: csipl_rscmmul_inter_f csipl_rscmmul_inter_d
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);	Computes the product, by element, of a real scalar and a complex matrix. The following instances are supported: csipl_rscmmul_split_f csipl_rscmmul_split_d
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);	Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported: csipl_vmmul_f csipl_vmmul_d
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);	Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported: csipl_cvmmul_inter_f csipl_cvmmul_inter_d
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);	Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported: csipl_cvmmul_split_f csipl_cvmmul_split_d

Prototype	Description
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);	Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported: csipl_rvcmmul_inter_f csipl_rvcmmul_inter_d
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);	Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported: csipl_rvcmmul_split_f csipl_rvcmmul_split_d
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);	Computes the difference, by element, of two vectors. The following instances are supported: csipl_vsub_f csipl_vsub_d csipl_vsub_i csipl_vsub_si
void csipl_cvsub_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);	Computes the difference, by element, of two vectors. The following instances are supported: csipl_cvsub_inter_f csipl_cvsub_inter_d
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);	Computes the difference, by element, of two vectors. The following instances are supported: csipl_cvsub_split_f csipl_cvsub_split_d

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

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

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

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

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

4.4 Ternary Operations

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

Prototype	Description
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);	Computes the sum of two vectors and product of a third vector, by element. The following instances are supported: csipl_cvam_split_f csipl_cvam_split_d csipl_cvma_split_f csipl_cvma_split_d
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);	Computes the product of two vectors and sum of a scalar, by element. The following instances are supported: csipl_vmsa_f csipl_vmsa_d
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);	Computes the product of two vectors and sum of a scalar, by element. The following instances are supported: csipl_cvmsa_inter_f csipl_cvmsa_inter_d
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);	Computes the product of two vectors and sum of a scalar, by element. The following instances are supported: csipl_cvmsa_split_f csipl_cvmsa_split_d
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);	Computes the product of two vectors and difference of a third vector, by element. The following instances are supported: csipl_vmsb_f csipl_vmsb_d

Prototype	Description
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);	Computes the product of two vectors and difference of a third vector, by element. The following instances are supported: csipl_cvmsb_inter_f csipl_cvmsb_inter_d
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);	Computes the product of two vectors and difference of a third vector, by element. The following instances are supported: csipl_cvmsb_split_f csipl_cvmsb_split_d
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);	Computes the sum of a vector and a scalar, and product with a second vector, by element. The following instances are supported: csipl_vsam_f csipl_vsam_d
void csipl_cvsam_inter_P(void * A, csipl_stride strideA, csipl_cscalar_P b, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);	Computes the sum of a vector and a scalar, and product with a second vector, by element. The following instances are supported: csipl_cvsam_inter_f csipl_cvsam_inter_d
void csipl_cvsam_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);	Computes the sum of a vector and a scalar, and product with a second vector, by element. The following instances are supported: csipl_cvsam_split_f csipl_cvsam_split_d

Prototype	Description
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);	Computes the difference of two vectors, and product with a third vector, by element. The following instances are supported: csipl_vsbm_f csipl_vsbm_d
void csipl_cvsbm_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);	Computes the difference of two vectors, and product with a third vector, by element. The following instances are supported: csipl_cvsbm_inter_f csipl_cvsbm_inter_d
void csipl_cvsbm_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);	Computes the difference of two vectors, and product with a third vector, by element. The following instances are supported: csipl_cvsbm_split_f csipl_cvsbm_split_d
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);	Computes the product of a vector and a scalar, and sum with a second vector, by element. The following instances are supported: csipl_vsma_f csipl_vsma_d
void csipl_cvsma_inter_P(void * A, csipl_stride strideA, csipl_cscalar_P b, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);	Computes the product of a vector and a scalar, and sum with a second vector, by element. The following instances are supported: csipl_cvsma_inter_f csipl_cvsma_inter_d

Prototype	Description
void csipl_cvsma_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);	Computes the product of a vector and a scalar, and sum with a second vector, by element. The following instances are supported: csipl_cvsma_split_f csipl_cvsma_split_d
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);	Computes the product of a vector and a scalar, and sum with a second scalar, by element. The following instances are supported: csipl_vsmsa_f csipl_vsmsa_d
void csipl_cvsmsa_inter_P(void * A, csipl_stride strideA, csipl_cscalar_P b, csipl_cscalar_P c, void * R, csipl_stride strideR, csipl_length n);	Computes the product of a vector and a scalar, and sum with a second scalar, by element. The following instances are supported: csipl_cvsmsa_inter_f csipl_cvsmsa_inter_d
void csipl_cvsmsa_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);	Computes the product of a vector and a scalar, and sum with a second scalar, by element. The following instances are supported: csipl_cvsmsa_split_f csipl_cvsmsa_split_d

4.5 Logical Operations

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

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

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

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

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

4.6 Selection Operations

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

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

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

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

4.7 Bitwise and Boolean Logical Operators

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

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

4.8 Element Generation and Copy

Prototype	Description
void csipl_vcopy_P_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);	Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types. The following instances are supported: 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
void csipl_cvcopy_inter_P_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);	Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types. The following instances are supported: csipl_cvcopy_inter_f_f csipl_cvcopy_inter_f_d csipl_cvcopy_inter_d_f csipl_cvcopy_inter_d_d
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);	Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types. The following instances are supported: csipl_cvcopy_split_f_f csipl_cvcopy_split_f_d csipl_cvcopy_split_d_f csipl_cvcopy_split_d_d

Prototype	Description
void csipl_mcopy_P_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);	Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types. The following instances are supported: 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
void csipl_cmcopy_inter_P_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);	Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types. The following instances are supported: csipl_cmcopy_inter_f_f csipl_cmcopy_inter_f_d csipl_cmcopy_inter_d_f csipl_cmcopy_inter_d_d
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);	Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types. The following instances are supported: csipl_cmcopy_split_f_f csipl_cmcopy_split_f_d csipl_cmcopy_split_d_f csipl_cmcopy_split_d_d
void csipl_vfill_P(scalar_P a, scalar_P * R, csipl_stride strideR, csipl_length n);	Fill a vector with a constant value. The following instances are supported: csipl_vfill_f csipl_vfill_d csipl_vfill_i csipl_vfill_si
void csipl_cvfill_inter_P(csipl_cscalar_P a, void * R, csipl_stride strideR, csipl_length n);	Fill a vector with a constant value. The following instances are supported: csipl_cvfill_inter_f csipl_cvfill_inter_d

Prototype	Description
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);	Fill a vector with a constant value. The following instances are supported: csipl_cvfill_split_f csipl_cvfill_split_d
void csipl_mfill_P(scalar_P a, scalar_P * R, int ldR, csipl_length m, csipl_length n);	Fill a matrix with a constant value. The following instances are supported: csipl_mfill_f csipl_mfill_d csipl_mfill_i csipl_mfill_si
void csipl_cmfill_inter_P(csipl_cscalar_P a, void * R, int ldR, csipl_length m, csipl_length n);	Fill a matrix with a constant value. The following instances are supported: csipl_cmfill_inter_f csipl_cmfill_inter_d
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);	Fill a matrix with a constant value. The following instances are supported: csipl_cmfill_split_f csipl_cmfill_split_d
void csipl_vramp_P(scalar_P alpha, scalar_P beta, scalar_P * R, csipl_stride strideR, csipl_length n);	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: csipl_vramp_f csipl_vramp_d csipl_vramp_i csipl_vramp_si

4.9 Manipulation Operations

Prototype	Description
void csipl_vcmplx_inter_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);	Form a complex vector from two real vectors. The following instances are supported: csipl_vcmplx_inter_f csipl_vcmplx_inter_d

Prototype	Description
void csipl_vcmplx_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);	Form a complex vector from two real vectors. The following instances are supported: csipl_vcmplx_split_f csipl_vcmplx_split_d
void csipl_mcmplx_P(scalar_P * A, int ldA, scalar_P * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);	Form a complex matrix from two real matrices. The following instances are supported: csipl_mcmplx_f csipl_mcmplx_d
void csipl_vgather_P(scalar_P * X, csipl_stride strideX, unsigned int * I, csipl_stride strideI, scalar_P * Y, csipl_stride strideY, csipl_length n);	The 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. The following instances are supported: csipl_vgather_f csipl_vgather_d csipl_vgather_i csipl_vgather_si
void csipl_cvgather_inter_P(void * X, csipl_stride strideX, unsigned int * I, csipl_stride strideI, void * Y, csipl_stride strideY, csipl_length n);	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. The following instances are supported: csipl_cvgather_inter_f csipl_cvgather_inter_d
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);	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. The following instances are supported: csipl_cvgather_split_f csipl_cvgather_split_d

Prototype	Description
void csipl_mgather_P(scalar_P * X, int ldX, csipl_scalar_mi * I, csipl_stride strideI, scalar_P * Y, csipl_stride strideY, csipl_length n);	The 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. The following instances are supported: csipl_mgather_f csipl_mgather_d csipl_mgather_i csipl_mgather_si
void csipl_cmgather_inter_P(void * X, int ldX, csipl_scalar_mi * I, csipl_stride strideI, void * Y, csipl_stride strideY, csipl_length n);	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. The following instances are supported: csipl_cmgather_inter_f csipl_cmgather_inter_d
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);	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. The following instances are supported: csipl_cmgather_split_f csipl_cmgather_split_d
void csipl_vimag_inter_P(void * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);	Extract the imaginary part of a complex vector. The following instances are supported: csipl_vimag_inter_f csipl_vimag_inter_d
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);	Extract the imaginary part of a complex vector. The following instances are supported: csipl_vimag_split_f csipl_vimag_split_d
void csipl_mimag_P(void * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);	Extract the imaginary part of a complex matrix. The following instances are supported: csipl_mimag_f csipl_mimag_d

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

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

Prototype	Description
void csipl_cvscatter_split_P(scalar_P * X_re, scalar_P * X_im, csipl_stride strideX, scalar_P * Y_re, scalar_P * Y_im, csipl_stride strideY, unsigned int * I_re, unsigned int * I_im, csipl_stride strideI, csipl_length n);	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: csipl_cvscatter_split_f csipl_cvscatter_split_d
void csipl_mscatter_P(scalar_P * X, csipl_stride strideX, scalar_P * Y, int ldY, csipl_scalar_mi * I, csipl_stride strideI, csipl_length n);	The 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. The following instances are supported: csipl_mscatter_f csipl_mscatter_d csipl_mscatter_i csipl_mscatter_si
void csipl_cmscatter_inter_P(void * X, csipl_stride strideX, void * Y, int ldY, csipl_scalar_mi * I, csipl_stride strideI, csipl_length n);	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. The following instances are supported: csipl_cmscatter_inter_f csipl_cmscatter_inter_d
void csipl_cmscatter_split_P(scalar_P * X_re, scalar_P * X_im, csipl_stride strideX, scalar_P * Y_re, scalar_P * Y_im, int ldY, csipl_scalar_mi * I_re, csipl_scalar_mi * I_im, csipl_stride strideI, csipl_length n);	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. The following instances are supported: csipl_cmscatter_split_f csipl_cmscatter_split_d
void csipl_cvswap_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_length n);	Swap elements between two vectors. The following instances are supported: csipl_cvswap_inter_f csipl_cvswap_inter_d

Prototype	Description
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);	Swap elements between two vectors. The following instances are supported: csipl_cvswap_split_f csipl_cvswap_split_d
void csipl_mswap_P(scalar_P * A, int ldA, scalar_P * B, int ldB, csipl_length m, csipl_length n);	Swap elements between two matrices. The following instances are supported: csipl_mswap_f csipl_mswap_d csipl_mswap_i csipl_mswap_si
void csipl_cmswap_inter_P(void * A, int ldA, void * B, int ldB, csipl_length m, csipl_length n);	Swap elements between two matrices. The following instances are supported: csipl_cmswap_inter_f csipl_cmswap_inter_d
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);	Swap elements between two matrices. The following instances are supported: csipl_cmswap_split_f csipl_cmswap_split_d

Chapter 5. Signal Processing Functions

5.1 FFT Functions

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

Prototype	Description
void csipl_ccfftop_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);	Apply a complex-to-complex Fast Fourier Transform (FFT). The following instances are supported: csipl_ccfftop_split_f csipl_ccfftop_split_d
void csipl_crfftop_inter_P(csipl_fft_P * plan, void * x, csipl_stride stridex, scalar_P * y, csipl_stride stridey);	Apply a complex-to-real Fast Fourier Transform (FFT). The following instances are supported: csipl_crfftop_inter_f csipl_crfftop_inter_d
void csipl_crfftop_split_P(csipl_fft_P * plan, scalar_P * x_re, scalar_P * x_im, csipl_stride stridex, scalar_P * y, csipl_stride stridey);	Apply a complex-to-real Fast Fourier Transform (FFT). The following instances are supported: csipl_crfftop_split_f csipl_crfftop_split_d
void csipl_rcfftop_inter_P(csipl_fft_P * plan, scalar_P * x, csipl_stride stridex, void * y, csipl_stride stridey);	Apply a real-to-complex Fast Fourier Transform (FFT). The following instances are supported: csipl_rcfftop_inter_f csipl_rcfftop_inter_d
void csipl_rcfftop_split_P(csipl_fft_P * plan, scalar_P * x, csipl_stride stridex, scalar_P * y_re, scalar_P * y_im, csipl_stride stridey);	Apply a real-to-complex Fast Fourier Transform (FFT). The following instances are supported: csipl_rcfftop_split_f csipl_rcfftop_split_d
int csipl_fft_destroy_P(csipl_fft_P * plan);	Destroy an FFT object. The following instances are supported: csipl_fft_destroy_f csipl_fft_destroy_d
void csipl_fft_getattr_P(csipl_fft_P * plan, csipl_fft_attr_P * attr);	Return the attributes of an FFT object. The following instances are supported: csipl_fft_getattr_f csipl_fft_getattr_d
*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);	Create a 1D multiple FFT object. The following instances are supported: csipl_ccfftmop_create_f csipl_ccfftmop_create_d

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

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

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

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

5.2 Convolution/Correlation Functions

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

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

Prototype	Description
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);	Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported: csipl_correlate1d_f csipl_correlate1d_d
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);	Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported: csipl_ccorrelate1d_inter_f csipl_ccorrelate1d_inter_d
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);	Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported: csipl_ccorrelate1d_split_f csipl_ccorrelate1d_split_d
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);	Create a 2D correlation object. The following instances are supported: csipl_corr2d_create_f csipl_corr2d_create_d csipl_ccorr2d_create_f csipl_ccorr2d_create_d
int csipl_Dcorr2d_destroy_P(csipl_Dcorr2d_P * plan);	Destroy a 2D correlation object. The following instances are supported: csipl_corr2d_destroy_f csipl_corr2d_destroy_d csipl_ccorr2d_destroy_f csipl_ccorr2d_destroy_d
void csipl_Dcorr2d_getattr_P(csipl_Dcorr2d_P * plan, csipl_Dcorr2d_attr_P * attr);	Return the attributes for a 2D correlation object. The following instances are supported: csipl_corr2d_getattr_f csipl_corr2d_getattr_d csipl_ccorr2d_getattr_f csipl_ccorr2d_getattr_d

Prototype	Description
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);	Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported: csipl_correlate2d_f csipl_correlate2d_d
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);	Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported: csipl_ccorrelate2d_inter_f csipl_ccorrelate2d_inter_d
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);	Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported: csipl_ccorrelate2d_split_f csipl_ccorrelate2d_split_d

5.3 Window Functions

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

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

5.4 Filter Functions

Prototype	Description
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);	Create a decimated FIR filter object. The following instances are supported: csipl_fir_create_f csipl_fir_create_d
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);	Create a decimated FIR filter object. The following instances are supported: csipl_cfir_create_inter_f csipl_cfir_create_inter_d
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);	Create a decimated FIR filter object. The following instances are supported: csipl_cfir_create_split_f csipl_cfir_create_split_d
int csipl_Dfir_destroy_P(csipl_Dfir_P * plan);	Destroy a FIR filter object. The following instances are supported: csipl_fir_destroy_f csipl_fir_destroy_d csipl_cfir_destroy_f csipl_cfir_destroy_d

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

5.5 Miscellaneous Signal Processing Functions

Prototype	Description
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);	Compute the histogram of a vector. The following instances are supported: csipl_vhisto_f csipl_vhisto_d

Chapter 6. Linear Algebra

6.1 Matrix and Vector Operations

Prototype	Description
void csipl_cmherm_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);	Complex Hermitian (conjugate transpose) of a matrix. The following instances are supported: csipl_cmherm_inter_f csipl_cmherm_inter_d
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);	Complex Hermitian (conjugate transpose) of a matrix. The following instances are supported: csipl_cmherm_split_f csipl_cmherm_split_d
csipl_cscalar_P csipl_cvjdot_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_length n);	Compute the conjugate inner (dot) product of two complex vectors. The following instances are supported: csipl_cvjdot_inter_f csipl_cvjdot_inter_d
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);	Compute the conjugate inner (dot) product of two complex vectors. The following instances are supported: csipl_cvjdot_split_f csipl_cvjdot_split_d
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);	Calculate the general product of two matrices and accumulate. The following instances are supported: csipl_gemp_f csipl_gemp_d

Prototype	Description
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);	Calculate the general product of two matrices and accumulate. The following instances are supported: csipl_cgemp_inter_f csipl_cgemp_inter_d
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);	Calculate the general product of two matrices and accumulate. The following instances are supported: csipl_cgemp_split_f csipl_cgemp_split_d
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);	Calculate a general matrix sum. The following instances are supported: csipl_gems_f csipl_gems_d
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);	Calculate a general matrix sum. The following instances are supported: csipl_cgems_inter_f csipl_cgems_inter_d

Prototype	Description
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);	Calculate a general matrix sum. The following instances are supported: csipl_cgems_split_f csipl_cgems_split_d
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);	Calculate the product of two matrices. The following instances are supported: csipl_mprod_f csipl_mprod_d csipl_mprod_i csipl_mprod_si
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);	Calculate the product of two matrices. The following instances are supported: csipl_cmprod_inter_f csipl_cmprod_inter_d csipl_cmprod_inter_i csipl_cmprod_inter_si
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);	Calculate the product of two matrices. The following instances are supported: csipl_cmprod_split_f csipl_cmprod_split_d csipl_cmprod_split_i csipl_cmprod_split_si
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);	Calculate the product a complex matrix and the Hermitian of a complex matrix. The following instances are supported: csipl_cmprodh_inter_f csipl_cmprodh_inter_d csipl_cmprodh_inter_i csipl_cmprodh_inter_si

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

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

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

Prototype	Description
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);	Calculate a vector–matrix product. The following instances are supported: csipl_cvmprod_split_f csipl_cvmprod_split_d csipl_cvmprod_split_i csipl_cvmprod_split_si
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);	Calculate the outer product of two vectors. The following instances are supported: csipl_vouter_f csipl_vouter_d
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);	Calculate the outer product of two vectors. The following instances are supported: csipl_cvouter_inter_f csipl_cvouter_inter_d
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);	Calculate the outer product of two vectors. The following instances are supported: csipl_cvouter_split_f csipl_cvouter_split_d
scalar_P csipl_vcsummgval_inter_P(void * A, csipl_stride strideA, csipl_length n);	Returns the sum of the magnitudes of the elements of a complex vector. The following instances are supported: csipl_vcsummgval_inter_f csipl_vcsummgval_inter_d
scalar_P csipl_vcsummgval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);	Returns the sum of the magnitudes of the elements of a complex vector. The following instances are supported: csipl_vcsummgval_split_f csipl_vcsummgval_split_d

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

6.2 Special Linear System Solvers

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

Prototype	Description
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);	Solve a real symmetric positive definite Toeplitz linear system. The following instances are supported: csipl_toepsol_f csipl_toepsol_d
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);	Solve a real symmetric positive definite Toeplitz linear system. The following instances are supported: csipl_ctoepsol_inter_f csipl_ctoepsol_inter_d
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);	Solve a real symmetric positive definite Toeplitz linear system. The following instances are supported: csipl_ctoepsol_split_f csipl_ctoepsol_split_d

6.3 General Square Linear System Solver

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

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

6.4 Symmetric Positive Definite Linear System Solver

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

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

Prototype	Description
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);	Solve a symmetric positive definite linear system. The following instances are supported: csipl_ccholsol_split_f csipl_ccholsol_split_d

6.5 Overdetermined Linear System Solver

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

Prototype	Description
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);	Multiply a matrix by the matrix Q from a QR decomposition. The following instances are supported: csipl_qrdprodq_f csipl_qrdprodq_d
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);	Multiply a matrix by the matrix Q from a QR decomposition. The following instances are supported: csipl_cqrdprodq_inter_f csipl_cqrdprodq_inter_d
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);	Multiply a matrix by the matrix Q from a QR decomposition. The following instances are supported: csipl_cqrdprodq_split_f csipl_cqrdprodq_split_d
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);	Solve linear system based on the matrix R, from QR decomposition of the matrix A. The following instances are supported: csipl_qrdsolr_f csipl_qrdsolr_d
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);	Solve linear system based on the matrix R, from QR decomposition of the matrix A. The following instances are supported: csipl_cqrdsolr_inter_f csipl_cqrdsolr_inter_d
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);	Solve linear system based on the matrix R, from QR decomposition of the matrix A. The following instances are supported: csipl_cqrdsolr_split_f csipl_cqrdsolr_split_d

Prototype	Description
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);	Solve linear system based on the matrix R, from QR decomposition of the matrix A. The following instances are supported: csipl_qrsol_f csipl_qrsol_d
int csipl_cqrsol_inter_P(csipl_cqr_P * qrd, csipl_qrd_prob prob, void * XB, int ldXB, csipl_length m, csipl_length n);	Solve either a linear covariance or linear least squares problem. The following instances are supported: csipl_cqrsol_inter_f csipl_cqrsol_inter_d
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);	Solve either a linear covariance or linear least squares problem. The following instances are supported: csipl_cqrsol_split_f csipl_cqrsol_split_d