



NASoftware Limited
Incorporating InfoSAR

CSIPL List of Contents

CSIPL/Contents [3.1]

Release 3.1
May 2013

Chapter 1. Support Functions

1.1 Initialisation and Finalisation

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

1.2 Sundry Functions

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

Chapter 2. Scalar Functions

2.1 Real Scalar Functions

Prototype	Description
<code>float csipl_acos_f(float A);</code>	Computes the principal radian value in $[0, \pi]$ of the inverse cosine of a scalar.
<code>float csipl_asin_f(float A);</code>	Computes the principal radian value in $[0, \pi]$ of the inverse sine of a scalar.
<code>float csipl_atan_f(float A);</code>	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent of a scalar.
<code>float csipl_atan2_f(float A, float B);</code>	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of two scalars.
<code>float csipl_ceil_f(float A);</code>	Computes the ceiling of a scalar.
<code>float csipl_cos_f(float A);</code>	Computes the cosine of a scalar angle in radians.
<code>float csipl_cosh_f(float A);</code>	Computes the hyperbolic cosine of a scalar.
<code>float csipl_exp_f(float A);</code>	Computes the exponential of a scalar.
<code>float csipl_floor_f(float A);</code>	Computes the floor of a scalar.
<code>float csipl_log_f(float A);</code>	Computes the natural logarithm of a scalar.
<code>float csipl_log10_f(float A);</code>	Computes the base 10 logarithm of a scalar.
<code>float csipl_mag_f(float A);</code>	Computes the magnitude (absolute value) of a scalar.
<code>float csipl_pow_f(float A, float B);</code>	Computes the power function of two scalars.
<code>float csipl_sin_f(float A);</code>	Computes the sine of a scalar angle in radians.
<code>float csipl_sinh_f(float A);</code>	Computes the hyperbolic sine of a scalar.
<code>float csipl_sqrt_f(float A);</code>	Computes the square root of a scalar.
<code>float csipl_tan_f(float A);</code>	Computes the tangent of a scalar angle in radians.

Prototype	Description
<code>float csipl_tanh_f(float A);</code>	Computes the hyperbolic tangent of a scalar.

2.2 Complex Scalar Functions

Prototype	Description
<code>float csipl_arg_f(csipl_cscalar_f x);</code>	Returns the argument in radians $[-\pi, \pi]$ of a complex scalar.
<code>csipl_cscalar_f csipl_cadd_f(csipl_cscalar_f x, csipl_cscalar_f y);</code>	Computes the complex sum of two scalars.
<code>csipl_cscalar_f csipl_rcadd_f(float x, csipl_cscalar_f y);</code>	Computes the complex sum of two scalars.
<code>csipl_cscalar_f csipl_cdiv_f(csipl_cscalar_f x, csipl_cscalar_f y);</code>	Computes the complex quotient of two scalars.
<code>csipl_cscalar_f csipl_crdiv_f(csipl_cscalar_f x, float y);</code>	Computes the complex quotient of two scalars.
<code>csipl_cscalar_f csipl_cexp_f(csipl_cscalar_f A);</code>	Computes the exponential of a scalar.
<code>csipl_cscalar_f csipl_cjmul_f(csipl_cscalar_f x, csipl_cscalar_f y);</code>	Computes the product a complex scalar with the conjugate of a second complex scalar.
<code>csipl_cscalar_f csipl_cmag_f(csipl_cscalar_f A);</code>	Computes the magnitude (absolute value) of a scalar.
<code>float csipl_cmagsq_f(csipl_cscalar_f x);</code>	Computes the magnitude squared of a complex scalar.
<code>csipl_cscalar_f csipl_cplx_f(float re, float im);</code>	Form a complex scalar from two real scalars.
<code>csipl_cscalar_f csipl_cmul_f(csipl_cscalar_f x, csipl_cscalar_f y);</code>	Computes the complex product of two scalars.
<code>csipl_cscalar_f csipl_rcmul_f(float x, csipl_cscalar_f y);</code>	Computes the complex product of two scalars.
<code>csipl_cscalar_f csipl_cneg_f(csipl_cscalar_f x);</code>	Computes the negation of a complex scalar.
<code>csipl_cscalar_f csipl_conj_f(csipl_cscalar_f x);</code>	Computes the complex conjugate of a scalar.
<code>csipl_cscalar_f csipl_crecip_f(csipl_cscalar_f x);</code>	Computes the reciprocal of a complex scalar.

Prototype	Description
<code>csipl_cscalar_f</code> <code>csipl_csqrt_f</code> (<code>csipl_cscalar_f x</code>);	Computes the square root a complex scalar.
<code>csipl_cscalar_f</code> <code>csipl_csub_f</code> (<code>csipl_cscalar_f x</code> , <code>csipl_cscalar_f y</code>);	Computes the complex difference of two scalars.
<code>csipl_cscalar_f</code> <code>csipl_rcsub_f</code> (<code>float x</code> , <code>csipl_cscalar_f y</code>);	Computes the complex difference of two scalars.
<code>csipl_cscalar_f</code> <code>csipl_crsub_f</code> (<code>csipl_cscalar_f x</code> , <code>float y</code>);	Computes the complex difference of two scalars.
<code>float</code> <code>csipl_imag_f</code> (<code>csipl_cscalar_f x</code>);	Extract the imaginary part of a complex scalar.
<code>void</code> <code>csipl_polar_f</code> (<code>csipl_cscalar_f a</code> , <code>float * r</code> , <code>float * t</code>);	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.
<code>float</code> <code>csipl_real_f</code> (<code>csipl_cscalar_f x</code>);	Extract the real part of a complex scalar.
<code>csipl_cscalar_f</code> <code>csipl_rect_f</code> (<code>float r</code> , <code>float t</code>);	Convert a pair of real scalars from complex polar to complex rectangular form.

2.3 Index Scalar Functions

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

Chapter 3. Random Number Generation

3.1 Random Number Functions

Prototype	Description
<pre><code>csipl_randstate * csipl_randcreate(csipl_index seed, csipl_index numprocs, csipl_index id, csipl_rng portable);</code></pre>	Create a random number generator state object.
<pre><code>int csipl_randdestroy(csipl_randstate * rand);</code></pre>	Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure.
<pre><code>float csipl_randu_f(csipl_randstate * state);</code></pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre><code>csipl_cscalar_f csipl_crandu_f(csipl_randstate * state);</code></pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre><code>void csipl_vrandu_f(csipl_randstate * state, float * R, csipl_stride strideR, csipl_length n);</code></pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre><code>void csipl_cvrandu_inter_f(csipl_randstate * state, void * R, csipl_stride strideR, csipl_length n);</code></pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre><code>void csipl_cvrandu_split_f(csipl_randstate * state, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</code></pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre><code>float csipl_randn_f(csipl_randstate * state);</code></pre>	Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.

Prototype	Description
<pre> csipl_cscalar_f csipl_crandn_f(csipl_randstate * state); </pre>	<p>Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.</p>
<pre> void csipl_vrandn_f(csipl_randstate * state, float * R, csipl_stride strideR, csipl_length n); </pre>	<p>Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.</p>
<pre> void csipl_cvrandn_inter_f(csipl_randstate * state, void * R, csipl_stride strideR, csipl_length n); </pre>	<p>Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.</p>
<pre> void csipl_cvrandn_split_f(csipl_randstate * state, float * R_re, float * R_im, csipl_stride strideR, csipl_length n); </pre>	<p>Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.</p>

Chapter 4. Vector And Elementwise Operations

4.1 Elementary Mathematical Functions

Prototype	Description
<pre>void csipl_vacos_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a vector.
<pre>void csipl_macos_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a matrix.
<pre>void csipl_vasin_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a vector.
<pre>void csipl_masin_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a matrix.
<pre>void csipl_vatan_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a vector.
<pre>void csipl_matan_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a matrix.
<pre>void csipl_vatan2_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input vectors.

Prototype	Description
<pre>void csipl_matan2_f(float * A, int ldA, float * B, int ldB, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input matrices.
<pre>void csipl_vcos_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the cosine for each element of a vector. Element angle values are in radians.
<pre>void csipl_mcos_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the cosine for each element of a matrix. Element angle values are in radians.
<pre>void csipl_vcosh_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the hyperbolic cosine for each element of a vector.
<pre>void csipl_mcosh_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the hyperbolic cosine for each element of a matrix.
<pre>void csipl_vexp_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the exponential function value for each element of a vector.
<pre>void csipl_cvexp_inter_f(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the exponential function value for each element of a vector.

Prototype	Description
<pre>void csipl_cvexp_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the exponential function value for each element of a vector.
<pre>void csipl_mexp_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the exponential function value for each element of a matrix.
<pre>void csipl_cmexp_inter_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the exponential function value for each element of a matrix.
<pre>void csipl_cmexp_split_f(float * A_re, float * A_im, int ldA, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the exponential function value for each element of a matrix.
<pre>void csipl_vexp10_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the base 10 exponential for each element of a vector.
<pre>void csipl_mexp10_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the base 10 exponential for each element of a matrix.
<pre>void csipl_vfloor_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the floor for each element of a vector.

Prototype	Description
<pre>void csipl_vlog_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the natural logarithm for each element of a vector.
<pre>void csipl_cvlog_inter_f(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the natural logarithm for each element of a vector.
<pre>void csipl_cvlog_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the natural logarithm for each element of a vector.
<pre>void csipl_mlog_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the natural logarithm for each element of a matrix.
<pre>void csipl_cmlog_inter_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the natural logarithm for each element of a matrix.
<pre>void csipl_cmlog_split_f(float * A_re, float * A_im, int ldA, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the natural logarithm for each element of a matrix.
<pre>void csipl_vlog10_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Compute the base ten logarithm for each element of a vector.

Prototype	Description
<pre>void csipl_mlog10_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the base ten logarithm for each element of a matrix.</p>
<pre>void csipl_vsin_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the sine for each element of a vector. Element angle values are in radians.</p>
<pre>void csipl_msin_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the sine for each element of a matrix. Element angle values are in radians.</p>
<pre>void csipl_vsinh_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the hyperbolic sine for each element of a vector.</p>
<pre>void csipl_msinh_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the hyperbolic sine for each element of a matrix.</p>
<pre>void csipl_vsqrt_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the square root for each element of a vector.</p>
<pre>void csipl_cvsqrt_inter_f(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the square root for each element of a vector.</p>
<pre>void csipl_cvsqrt_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the square root for each element of a vector.</p>

Prototype	Description
<pre>void csipl_msqrt_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Compute the square root for each element of a matrix.
<pre>void csipl_cmsqrt_inter_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Compute the square root for each element of a matrix.
<pre>void csipl_cmsqrt_split_f(float * A_re, float * A_im, int ldA, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Compute the square root for each element of a matrix.
<pre>void csipl_mtanh_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Compute the tangent for each element of a matrix. Element angle values are in radians.
<pre>void csipl_mtanh_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the hyperbolic tangent for each element of a matrix.

4.2 Unary Operations

Prototype	Description
<pre>void csipl_varg_f(void * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the argument in radians $[-\pi, \pi]$ for each element of a complex vector.
<pre>void csipl_marg_f(void * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the argument in radians $[-\pi, \pi]$ for each element of a complex matrix.

Prototype	Description
<pre>void csipl_vceil_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the ceiling for each element of a vector.
<pre>void csipl_cvconj_inter_f(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	Compute the conjugate for each element of a complex vector.
<pre>void csipl_cvconj_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Compute the conjugate for each element of a complex vector.
<pre>void csipl_cmconj_inter_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Compute the conjugate for each element of a complex matrix.
<pre>void csipl_cmconj_split_f(float * A_re, float * A_im, int ldA, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Compute the conjugate for each element of a complex matrix.
<pre>void csipl_vcumsum_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	Compute the cumulative sum of the elements of a vector. The following instances are supported: <code>csipl_vcumsum_f</code> <code>csipl_vcumsum_i</code>
<pre>void csipl_cvcumsum_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	Compute the cumulative sum of the elements of a vector. The following instances are supported: <code>csipl_cvcumsum_inter_f</code> <code>csipl_cvcumsum_inter_i</code>

Prototype	Description
<pre>void csipl_cvcumsum_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the cumulative sum of the elements of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cvcumsum_split_f</code> <code>csipl_cvcumsum_split_i</code></p>
<pre>void csipl_mcumsum_P(csipl_major dir, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Compute the cumulative sums of the elements in the rows or columns of a matrix.</p> <p>The following instances are supported:</p> <p><code>csipl_mcumsum_f</code> <code>csipl_mcumsum_i</code></p>
<pre>void csipl_cmcumsum_inter_P(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the cumulative sum of the elements of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cmcumsum_inter_f</code> <code>csipl_cmcumsum_inter_i</code></p>
<pre>void csipl_cmcumsum_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the cumulative sum of the elements of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_cmcumsum_split_f</code> <code>csipl_cmcumsum_split_i</code></p>
<pre>void csipl_veuler_f(float * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a vector.</p>
<pre>void csipl_meuler_f(float * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a matrix.</p>
<pre>void csipl_vmag_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the magnitude for each element of a vector.</p> <p>The following instances are supported:</p> <p><code>csipl_vmag_f</code> <code>csipl_vmag_i</code> <code>csipl_vmag_si</code></p>
<pre>void csipl_cvmag_inter_f(void * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Compute the magnitude for each element of a vector.</p>

Prototype	Description
<pre>void csipl_cvmag_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Compute the magnitude for each element of a vector.
<pre>void csipl_mmag_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Compute the magnitude for each element of a matrix.
<pre>void csipl_cmmag_inter_f(void * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Compute the magnitude for each element of a matrix.
<pre>void csipl_cmmag_split_f(float * A_re, float * A_im, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Compute the magnitude for each element of a matrix.
<pre>void csipl_vcmsgsq_inter_f(void * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the square of the magnitudes for each element of a vector.
<pre>void csipl_vcmsgsq_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the square of the magnitudes for each element of a vector.
<pre>void csipl_mcmagsq_f(void * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the square of the magnitudes for each element of a matrix.
<pre>float csipl_vmeanval_f(float * A, csipl_stride strideA, csipl_length n);</pre>	Returns the mean value of the elements of a vector.

Prototype	Description
<code>csipl_cscalar_f csipl_cvmeanval_inter_f(void * A, csipl_stride strideA, csipl_length n);</code>	Returns the mean value of the elements of a vector.
<code>csipl_cscalar_f csipl_cvmeanval_split_f(float * A_re, float * A_im, csipl_stride strideA, csipl_length n);</code>	Returns the mean value of the elements of a vector.
<code>float csipl_mmeanval_f(float * A, int ldA, csipl_length m, csipl_length n);</code>	Returns the mean value of the elements of a matrix.
<code>csipl_cscalar_f csipl_cmmeanval_inter_f(void * A, int ldA, csipl_length m, csipl_length n);</code>	Returns the mean value of the elements of a matrix.
<code>csipl_cscalar_f csipl_cmmeanval_split_f(float * A_re, float * A_im, int ldA, csipl_length m, csipl_length n);</code>	Returns the mean value of the elements of a matrix.
<code>float csipl_vmeansqval_f(float * A, csipl_stride strideA, csipl_length n);</code>	Returns the mean magnitude squared value of the elements of a vector.
<code>float csipl_cvmeansqval_inter_f(void * A, csipl_stride strideA, csipl_length n);</code>	Returns the mean magnitude squared value of the elements of a vector.
<code>float csipl_cvmeansqval_split_f(float * A_re, float * A_im, csipl_stride strideA, csipl_length n);</code>	Returns the mean magnitude squared value of the elements of a vector.
<code>float csipl_mmeansqval_f(float * A, int ldA, csipl_length m, csipl_length n);</code>	Returns the mean magnitude squared value of the elements of a matrix.
<code>float csipl_cmmeansqval_inter_f(void * A, int ldA, csipl_length m, csipl_length n);</code>	Returns the mean magnitude squared value of the elements of a matrix.

Prototype	Description
<pre>float csipl_cmmeansqval_split_f(float * A_re, float * A_im, int ldA, csipl_length m, csipl_length n);</pre>	Returns the mean magnitude squared value of the elements of a matrix.
<pre>float csipl_vmodulate_f(float * A, csipl_stride strideA, float nu, float phi, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the modulation of a real vector by a specified complex frequency.
<pre>float csipl_cvmodulate_inter_f(void * A, csipl_stride strideA, float nu, float phi, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the modulation of a real vector by a specified complex frequency.
<pre>float csipl_cvmodulate_split_f(float * A_re, float * A_im, csipl_stride strideA, float nu, float phi, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the modulation of a real vector by a specified complex frequency.
<pre>void csipl_vneg_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	Computes the negation for each element of a vector. The following instances are supported: <code>csipl_vneg_f</code> <code>csipl_vneg_i</code> <code>csipl_vneg_si</code>
<pre>void csipl_cvneg_inter_f(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the negation for each element of a vector.
<pre>void csipl_cvneg_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the negation for each element of a vector.

Prototype	Description
<pre>void csipl_mneg_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the negation for each element of a matrix.</p> <p>The following instances are supported:</p> <pre>csipl_mneg_f csipl_mneg_i</pre>
<pre>void csipl_cmneg_inter_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the negation for each element of a matrix.</p>
<pre>void csipl_cmneg_split_f(float * A_re, float * A_im, int ldA, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the negation for each element of a matrix.</p>
<pre>void csipl_vrecip_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a vector.</p>
<pre>void csipl_cvrecip_inter_f(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a vector.</p>
<pre>void csipl_cvrecip_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a vector.</p>
<pre>void csipl_mrecip_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the reciprocal for each element of a matrix.</p>

Prototype	Description
<pre>void csipl_cmrecip_inter_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the reciprocal for each element of a matrix.
<pre>void csipl_cmrecip_split_f(float * A_re, float * A_im, int ldA, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the reciprocal for each element of a matrix.
<pre>void csipl_vrsqrt_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the reciprocal of the square root for each element of a vector.
<pre>void csipl_mrsqrt_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the reciprocal of the square root for each element of a matrix.
<pre>void csipl_vsq_f(float * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the square for each element of a vector.
<pre>void csipl_msq_f(float * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the square for each element of a matrix.
<pre>scalar_P csipl_cvsumval_inter_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	Returns the sum of the elements of a vector. The following instances are supported: <code>csipl_cvsumval_inter_f</code> <code>csipl_cvsumval_inter_i</code>
<pre>scalar_P csipl_cvsumval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	Returns the sum of the elements of a vector. The following instances are supported: <code>csipl_cvsumval_split_f</code> <code>csipl_cvsumval_split_i</code>

Prototype	Description
<pre>scalar_P csipl_msumval_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector.</p> <p>The following instances are supported:</p> <pre>csipl_msumval_f csipl_msumval_i</pre>
<pre>scalar_P csipl_csumval_inter_P(scalar_P * A, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector.</p> <p>The following instances are supported:</p> <pre>csipl_csumval_inter_f csipl_csumval_inter_i</pre>
<pre>scalar_P csipl_csumval_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, csipl_length n);</pre>	<p>Returns the sum of the elements of a vector.</p> <p>The following instances are supported:</p> <pre>csipl_csumval_split_f csipl_csumval_split_i</pre>
<pre>float csipl_msumsqval_f(float * A, int ldA, csipl_length m, csipl_length n);</pre>	<p>Returns the sum of the squares of the elements of a matrix.</p>

4.3 Binary Operations

Prototype	Description
<pre>void csipl_vadd_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_vadd_f csipl_vadd_i csipl_vadd_si</pre>
<pre>void csipl_cvadd_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_cvadd_inter_f csipl_cvadd_inter_i</pre>
<pre>void csipl_cvadd_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>csipl_cvadd_split_f csipl_cvadd_split_i</pre>

Prototype	Description
<pre>void csipl_madd_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of two matrices.</p> <p>The following instances are supported:</p> <pre>csipl_madd_f csipl_madd_i</pre>
<pre>void csipl_cmadd_inter_f(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of two matrices.</p>
<pre>void csipl_cmadd_split_f(float * A_re, float * A_im, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of two matrices.</p>
<pre>void csipl_rcvadd_inter_f(float * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of two vectors.</p>
<pre>void csipl_rcvadd_split_f(float * A, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of two vectors.</p>
<pre>void csipl_rcmadd_inter_f(float * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of two matrices.</p>

Prototype	Description
<pre>void csipl_rcmadd_split_f(float * A, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of two matrices.</p>
<pre>void csipl_svadd_P(scalar_P a, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a vector. The following instances are supported:</p> <p><code>csipl_svadd_f</code> <code>csipl_svadd_i</code> <code>csipl_svadd_si</code></p>
<pre>void csipl_csvadd_inter_f(csipl_cscalar_f a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a vector.</p>
<pre>void csipl_csvadd_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a vector.</p>
<pre>void csipl_smadd_P(scalar_P a, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>csipl_smadd_f</code> <code>csipl_smadd_i</code></p>
<pre>void csipl_csmadd_inter_f(csipl_cscalar_f a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the sum, by element, of a scalar and a matrix.</p>

Prototype	Description
<pre>void csipl_csmadd_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the sum, by element, of a scalar and a matrix.
<pre>void csipl_rscvadd_inter_f(float a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the sum, by element, of a real scalar and a complex vector.
<pre>void csipl_rscvadd_split_f(float a, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the sum, by element, of a real scalar and a complex vector.
<pre>void csipl_rscmadd_inter_f(float a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the sum, by element, of a real scalar and a complex matrix.
<pre>void csipl_rscmadd_split_f(float a, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the sum, by element, of a real scalar and a complex matrix.
<pre>void csipl_Dvdiv_P(csipl_Dvview_P * A, csipl_stride strideA, csipl_Dvview_P * B, csipl_stride strideB, csipl_Dvview_P * R, csipl_stride strideR, csipl_length n);</pre>	Computes the quotient, by element, of two vectors. The following instances are supported: <code>csipl_vdiv_f</code> <code>csipl_svdiv_f</code>

Prototype	Description
<pre>void csipl_Dvdiv_inter_P(csipl_Dvview_P * A, csipl_stride strideA, csipl_Dvview_P * B, csipl_stride strideB, csipl_Dvview_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_crdiv_inter_f</code> <code>csipl_cvdiv_inter_f</code></p>
<pre>void csipl_Dvdiv_split_P(csipl_Dvview_P * A_re, csipl_Dvview_P * A_im, csipl_stride strideA, csipl_Dvview_P * B_re, csipl_Dvview_P * B_im, csipl_stride strideB, csipl_Dvview_P * R_re, csipl_Dvview_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_crdiv_split_f</code> <code>csipl_cvdiv_split_f</code></p>
<pre>void csipl_mdiv_f(float * A, int ldA, float * B, int ldB, float * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p>
<pre>void csipl_cmdiv_inter_f(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p>
<pre>void csipl_cmdiv_split_f(float * A_re, float * A_im, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p>
<pre>void csipl_rcvdiv_inter_f(float a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of two vectors.</p>

Prototype	Description
<pre>void csipl_rcvdiv_split_f(float a, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the quotient, by element, of two vectors.</p>
<pre>void csipl_rcmdiv_inter_f(float a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p>
<pre>void csipl_rcmdiv_split_f(float a, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p>
<pre>void csipl_crmdiv_inter_f(void * A, int ldA, float * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p>
<pre>void csipl_crmdiv_split_f(float * A_re, float * A_im, int ldA, float * B, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the quotient, by element, of two matrices.</p>
<pre>void csipl_rscmsub_inter_f(float a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of a real scalar and a complex matrix.</p>

Prototype	Description
<pre>void csipl_rscsub_split_f(float a, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the difference, by element, of a real scalar and a complex matrix.
<pre>void csipl_vsdiv_f(float * A, csipl_stride strideA, float b, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the quotient, by element, of a vector and a scalar.
<pre>void csipl_cvrdiv_inter_f(float * A, csipl_stride strideA, float b, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the quotient, by element, of a vector and a scalar.
<pre>void csipl_cvrdiv_split_f(float * A_re, float * A_im, csipl_stride strideA, float b_re, float b_im, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the quotient, by element, of a vector and a scalar.
<pre>void csipl_msdiv_f(float * A, int ldA, float b, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a matrix and a scalar.
<pre>void csipl_cmrsdiv_inter_f(void * A, int ldA, csipl_cscalar_f b, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a matrix and a scalar.

Prototype	Description
<pre>void csipl_cmrsdiv_split_f(float * A_re, float * A_im, int ldA, csipl_cscalar_f b_re, csipl_cscalar_f b_im, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a matrix and a scalar.
<pre>void csipl_vexpoavg_f(float a, float * B, csipl_stride strideB, float * C, csipl_stride strideC, csipl_length n);</pre>	Computes an exponential weighted average, by element, of two vectors.
<pre>void csipl_cvexpoavg_inter_f(float a, void * B, csipl_stride strideB, void * C, csipl_stride strideC, csipl_length n);</pre>	Computes an exponential weighted average, by element, of two vectors.
<pre>void csipl_cvexpoavg_split_f(float a, float * B_re, float * B_im, csipl_stride strideB, float * C_re, float * C_im, csipl_stride strideC, csipl_length n);</pre>	Computes an exponential weighted average, by element, of two vectors.
<pre>void csipl_mexpoavg_f(float a, float * B, int ldB, float * C, int ldC, csipl_length m, csipl_length n);</pre>	Computes an exponential weighted average, by element, of two matrices.
<pre>void csipl_cmexpoavg_inter_f(float a, void * B, int ldB, void * C, int ldC, csipl_length m, csipl_length n);</pre>	Computes an exponential weighted average, by element, of two matrices.

Prototype	Description
<pre>void csipl_cmexpoavg_split_f(float a, float * B_re, float * B_im, int ldB, float * C_re, float * C_im, int ldC, csipl_length m, csipl_length n);</pre>	Computes an exponential weighted average, by element, of two matrices.
<pre>void csipl_vhypot_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the square root of the sum of squares, by element, of two input vectors.
<pre>void csipl_mhypot_f(float * A, int ldA, float * B, int ldB, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the square root of the sum of squares, by element, of two input matrices.
<pre>void csipl_cvjmul_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the product of a complex vector with the conjugate of a second complex vector, by element.
<pre>void csipl_cvjmul_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the product of a complex vector with the conjugate of a second complex vector, by element.
<pre>void csipl_cmjmul_inter_f(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.

Prototype	Description
<pre>void csipl_cmjmul_split_f(float * A_re, float * A_im, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.</p>
<pre>void csipl_Dvmul_P(csipl_Dvview_P * A, csipl_stride strideA, csipl_Dvview_P * B, csipl_stride strideB, csipl_Dvview_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of two vectors. The following instances are supported:</p> <pre>csipl_vmul_f csipl_vmul_i csipl_vmul_si csipl_svmul_f csipl_svmul_i csipl_svmul_si</pre>
<pre>void csipl_cvmul_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of two vectors.</p>
<pre>void csipl_cvmul_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of two vectors.</p>
<pre>void csipl_mmul_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of two matrices. The following instances are supported:</p> <pre>csipl_mmul_f csipl_mmul_i</pre>

Prototype	Description
<pre>void csipl_cmmul_inter_f(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of two matrices.
<pre>void csipl_cmmul_split_f(float * A_re, float * A_im, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of two matrices.
<pre>void csipl_rcvmul_inter_f(float * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the product, by element, of two vectors.
<pre>void csipl_rcvmul_split_f(float * A, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the product, by element, of two vectors.
<pre>void csipl_rcmmul_inter_f(float * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of two matrices.

Prototype	Description
<pre>void csipl_rcmmul_split_f(float * A, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of two matrices.</p>
<pre>void csipl_rscvmul_inter_f(float a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of a real scalar and a complex vector.</p>
<pre>void csipl_rscvmul_split_f(float a, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of a real scalar and a complex vector.</p>
<pre>void csipl_csvmul_inter_f(csipl_cscalar_f a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of a scalar and a vector.</p>
<pre>void csipl_csvmul_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product, by element, of a scalar and a vector.</p>
<pre>void csipl_smmul_f(float a, float * B, int ldB, float * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a scalar and a matrix.</p>

Prototype	Description
<pre>void csipl_csmmul_inter_f(csipl_cscalar_f a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of a scalar and a matrix.
<pre>void csipl_csmmul_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of a scalar and a matrix.
<pre>void csipl_rscmmul_inter_f(float a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of a real scalar and a complex matrix.
<pre>void csipl_rscmmul_split_f(float a, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of a real scalar and a complex matrix.
<pre>void csipl_vmmul_f(float * A, csipl_stride strideA, float * B, int ldB, csipl_major major, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of a vector and the rows or columns of a matrix.
<pre>void csipl_cvmmul_inter_f(void * A, csipl_stride strideA, void * B, int ldB, csipl_major major, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype	Description
<pre>void csipl_cvmmul_split_f(csipl_Dvview_f * A_re, csipl_Dvview_f * A_im, csipl_stride strideA, csipl_Dmview_f * B_re, csipl_Dmview_f * B_im, int ldB, csipl_major major, csipl_Dmview_f * R_re, csipl_Dmview_f * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix.</p>
<pre>void csipl_rvcmmul_inter_f(float * A, csipl_stride strideA, void * B, int ldB, csipl_major major, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix.</p>
<pre>void csipl_rvcmmul_split_f(float * A, csipl_stride strideA, float * B_re, float * B_im, int ldB, csipl_major major, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix.</p>
<pre>void csipl_vsub_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors. The following instances are supported:</p> <pre>csipl_vsub_f csipl_vsub_i csipl_vsub_si</pre>
<pre>void csipl_cvsub_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors.</p>

Prototype	Description
<pre>void csipl_cvsub_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors.</p>
<pre>void csipl_msub_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices. The following instances are supported:</p> <pre>csipl_msub_f csipl_msub_i</pre>
<pre>void csipl_cmsub_inter_f(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices.</p>
<pre>void csipl_cmsub_split_f(float * A_re, float * A_im, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices.</p>
<pre>void csipl_crmsub_inter_f(void * A, int ldA, float * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of two matrices.</p>

Prototype	Description
<pre>void csipl_crmsub_split_f(float * A_re, float * A_im, int ldA, float * B, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the difference, by element, of two matrices.
<pre>void csipl_rcvsub_inter_f(float * A, csipl_stride strideA, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the difference, by element, of two vectors.
<pre>void csipl_rcvsub_split_f(float * A, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the difference, by element, of two vectors.
<pre>void csipl_rcmsub_inter_f(float * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the difference, by element, of two matrices.
<pre>void csipl_rcmsub_split_f(float * A, int ldA, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the difference, by element, of two matrices.
<pre>void csipl_crvsub_inter_f(void * A, csipl_stride strideA, float * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the difference, by element, of two vectors.

Prototype	Description
<pre>void csipl_crvsub_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of two vectors.</p>
<pre>void csipl_svsub_P(scalar_P a, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a vector. The following instances are supported:</p> <p>csipl_svsub_f csipl_svsub_i csipl_svsub_si</p>
<pre>void csipl_csvsub_inter_f(csipl_cscalar_f a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a vector.</p>
<pre>void csipl_csvsub_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a vector.</p>
<pre>void csipl_smsub_P(scalar_P a, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a matrix. The following instances are supported:</p> <p>csipl_smsub_f csipl_smsub_i</p>
<pre>void csipl_csmsub_inter_f(csipl_cscalar_f a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the difference, by element, of a scalar and a matrix.</p>

Prototype	Description
<pre>void csipl_csmsub_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the difference, by element, of a scalar and a matrix.
<pre>void csipl_smdiv_f(float a, float * B, int ldB, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a scalar and a matrix.
<pre>void csipl_csmdiv_inter_f(csipl_cscalar_f a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a scalar and a matrix.
<pre>void csipl_csmdiv_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a scalar and a matrix.
<pre>void csipl_rscvdiv_inter_f(float a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the quotient, by element, of a real scalar and a complex vector.
<pre>void csipl_rscvdiv_split_f(float a, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the quotient, by element, of a real scalar and a complex vector.

Prototype	Description
<pre>void csipl_rscvsub_inter_f(float a, void * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the difference, by element, of a real scalar and a complex vector.
<pre>void csipl_rscvsub_split_f(float a, float * B_re, float * B_im, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the difference, by element, of a real scalar and a complex vector.
<pre>void csipl_rscmdiv_inter_f(float a, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a real scalar and a complex matrix.
<pre>void csipl_rscmdiv_split_f(float a, float * B_re, float * B_im, int ldB, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Computes the quotient, by element, of a real scalar and a complex matrix.

4.4 Ternary Operations

Prototype	Description
<pre>void csipl_vam_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * C, csipl_stride strideC, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	Computes the sum of two vectors and product of a third vector, by element. The following instances are supported: <code>csipl_vam_f</code> <code>csipl_vma_f</code>

Prototype	Description
<pre>void csipl_cvam_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum of two vectors and product of a third vector, by element. The following instances are supported:</p> <pre>csipl_cvam_inter_f csipl_cvma_inter_f</pre>
<pre>void csipl_cvam_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, scalar_P * C_re, scalar_P * C_im, csipl_stride strideC, scalar_P * R_re, scalar_P * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum of two vectors and product of a third vector, by element. The following instances are supported:</p> <pre>csipl_cvam_split_f csipl_cvma_split_f</pre>
<pre>void csipl_vmsa_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float c, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and sum of a scalar, by element.</p>
<pre>void csipl_cvmsa_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_cscalar_f c, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and sum of a scalar, by element.</p>
<pre>void csipl_cvmsa_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, csipl_cscalar_f c_re, csipl_cscalar_f c_im, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and sum of a scalar, by element.</p>

Prototype	Description
<pre>void csipl_vmsb_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * C, csipl_stride strideC, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and difference of a third vector, by element.</p>
<pre>void csipl_cvmsb_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and difference of a third vector, by element.</p>
<pre>void csipl_cvmsb_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * C_re, float * C_im, csipl_stride strideC, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of two vectors and difference of a third vector, by element.</p>
<pre>void csipl_vsam_f(float * A, csipl_stride strideA, float b, float * C, csipl_stride strideC, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum of a vector and a scalar, and product with a second vector, by element.</p>
<pre>void csipl_cvsam_inter_f(void * A, csipl_stride strideA, csipl_cscalar_f b, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum of a vector and a scalar, and product with a second vector, by element.</p>

Prototype	Description
<pre>void csipl_cvsam_split_f(float * A_re, float * A_im, csipl_stride strideA, csipl_cscalar_f b_re, csipl_cscalar_f b_im, float * C_re, float * C_im, csipl_stride strideC, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the sum of a vector and a scalar, and product with a second vector, by element.</p>
<pre>void csipl_vsbm_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * C, csipl_stride strideC, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference of two vectors, and product with a third vector, by element.</p>
<pre>void csipl_cvsbm_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference of two vectors, and product with a third vector, by element.</p>
<pre>void csipl_cvsbm_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * C_re, float * C_im, csipl_stride strideC, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the difference of two vectors, and product with a third vector, by element.</p>
<pre>void csipl_vsma_f(float * A, csipl_stride strideA, float b, float * C, csipl_stride strideC, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second vector, by element.</p>

Prototype	Description
<pre>void csipl_cvmsa_inter_f(void * A, csipl_stride strideA, csipl_cscalar_f b, void * C, csipl_stride strideC, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the product of a vector and a scalar, and sum with a second vector, by element.
<pre>void csipl_cvmsa_split_f(float * A_re, float * A_im, csipl_stride strideA, csipl_cscalar_f b_re, csipl_cscalar_f b_im, float * C_re, float * C_im, csipl_stride strideC, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the product of a vector and a scalar, and sum with a second vector, by element.
<pre>void csipl_vmsa_f(float * A, csipl_stride strideA, float b, float c, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the product of a vector and a scalar, and sum with a second scalar, by element.
<pre>void csipl_cvmsa_inter_f(void * A, csipl_stride strideA, csipl_cscalar_f b, csipl_cscalar_f c, void * R, csipl_stride strideR, csipl_length n);</pre>	Computes the product of a vector and a scalar, and sum with a second scalar, by element.
<pre>void csipl_cvmsa_split_f(float * A_re, float * A_im, csipl_stride strideA, csipl_cscalar_f b_re, csipl_cscalar_f b_im, csipl_cscalar_f c_re, csipl_cscalar_f c_im, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Computes the product of a vector and a scalar, and sum with a second scalar, by element.

4.5 Logical Operations

Prototype	Description
<code>csipl_scalar_bl csipl_valltrue_bl(signed int * A, csipl_stride strideA, csipl_length n);</code>	Returns true if all the elements of a vector are true.
<code>csipl_scalar_bl csipl_malltrue_bl(signed int * A, csipl_stride strideA, csipl_length n);</code>	Returns true if all the elements of a vector are true.
<code>csipl_scalar_bl csipl_vanytrue_bl(signed int * A, csipl_stride strideA, csipl_length n);</code>	Returns true if one or more elements of a vector are true.
<code>csipl_scalar_bl csipl_manytrue_bl(signed int * A, csipl_stride strideA, csipl_length n);</code>	Returns true if one or more elements of a vector are true.
<code>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);</code>	Computes the boolean comparison of 'equal', by element, of two vectors. The following instances are supported: <code>csipl_vleq_f</code> <code>csipl_vleq_i</code>
<code>void csipl_cvleq_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</code>	Computes the boolean comparison of 'equal', by element, of two vectors. The following instances are supported: <code>csipl_cvleq_inter_f</code> <code>csipl_cvleq_inter_i</code>
<code>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);</code>	Computes the boolean comparison of 'equal', by element, of two vectors. The following instances are supported: <code>csipl_cvleq_split_f</code> <code>csipl_cvleq_split_i</code>
<code>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);</code>	Computes the boolean comparison of 'equal', by element, of two vectors/matrices. The following instances are supported: <code>csipl_mleq_f</code> <code>csipl_mleq_i</code>

Prototype	Description
<pre>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);</pre>	<p>Computes the boolean comparison of ‘equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_cmleq_inter_f</code> <code>csipl_cmleq_inter_i</code></p>
<pre>void csipl_cmleq_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B_re, scalar_P * B_im, int ldB, signed int * R_re, signed int * R_im, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_cmleq_split_f</code> <code>csipl_cmleq_split_i</code></p>
<pre>void csipl_vlge_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘greater than or equal’, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vlge_f</code> <code>csipl_vlge_i</code></p>
<pre>void csipl_mlge_P(scalar_P * A, int ldA, scalar_P * B, int ldB, signed int * R, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘greater than or equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_mlge_f</code> <code>csipl_mlge_i</code></p>
<pre>void csipl_vlgt_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘greater than’, by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vlgt_f</code> <code>csipl_vlgt_i</code></p>
<pre>void csipl_mlgt_P(scalar_P * A, int ldA, scalar_P * B, int ldB, signed int * R, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘greater than’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_mlgt_f</code> <code>csipl_mlgt_i</code></p>

Prototype	Description
<pre>void csipl_vlle_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of 'less than or equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vlle_f</code> <code>csipl_vlle_i</code></p>
<pre>void csipl_mlle_P(scalar_P * A, int ldA, scalar_P * B, int ldB, signed int * R, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of 'less than or equal', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_mlle_f</code> <code>csipl_mlle_i</code></p>
<pre>void csipl_vllt_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of 'less than', by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vllt_f</code> <code>csipl_vllt_i</code></p>
<pre>void csipl_mllt_P(scalar_P * A, int ldA, scalar_P * B, int ldB, signed int * R, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of 'less than', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p><code>csipl_mllt_f</code> <code>csipl_mllt_i</code></p>
<pre>void csipl_vlne_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of 'not equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_vlne_f</code> <code>csipl_vlne_i</code></p>
<pre>void csipl_cvlne_inter_P(void * A, csipl_stride strideA, void * B, csipl_stride strideB, signed int * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of 'not equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <p><code>csipl_cvlne_inter_f</code> <code>csipl_cvlne_inter_i</code></p>

Prototype	Description
<pre>void csipl_cvlne_split_P(scalar_P * A_re, scalar_P * A_im, csipl_stride strideA, scalar_P * B_re, scalar_P * B_im, csipl_stride strideB, signed int * R_re, signed int * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_cvlne_split_f</code> <code>csipl_cvlne_split_i</code></p>
<pre>void csipl_mlne_P(scalar_P * A, int ldA, scalar_P * B, int ldB, signed int * R, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors/matrices. The following instances are supported:</p> <p><code>csipl_mlne_f</code> <code>csipl_mlne_i</code></p>
<pre>void csipl_cmlne_inter_P(void * A, int ldA, void * B, int ldB, signed int * R, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors/matrices. The following instances are supported:</p> <p><code>csipl_cmlne_inter_f</code> <code>csipl_cmlne_inter_i</code></p>
<pre>void csipl_cmlne_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B_re, scalar_P * B_im, int ldB, signed int * R_re, signed int * R_im, csipl_stride strideR, csipl_length m, csipl_length n);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors/matrices. The following instances are supported:</p> <p><code>csipl_cmlne_split_f</code> <code>csipl_cmlne_split_i</code></p>

4.6 Selection Operations

Prototype	Description
<pre>void csipl_vclip_P(scalar_P * A, csipl_stride strideA, scalar_P t1, scalar_P t2, scalar_P c1, scalar_P c2, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the generalised double clip, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_vclip_f</code> <code>csipl_vclip_i</code> <code>csipl_vclip_si</code></p>

Prototype	Description
<pre>void csipl_vinvclip_P(scalar_P * A, csipl_stride strideA, scalar_P t1, scalar_P t2, scalar_P t3, scalar_P c1, scalar_P c2, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the generalised inverted double clip, by element, of two vectors. The following instances are supported:</p> <p><code>csipl_vinvclip_f</code> <code>csipl_vinvclip_i</code> <code>csipl_vinvclip_si</code></p>
<pre>unsigned int csipl_vindexbool(signed int * X, csipl_stride strideX, unsigned int * Y, csipl_stride strideY, csipl_length n);</pre>	<p>Computes an index vector of the indices of the non-false elements of the boolean vector, and returns the number of non-false elements.</p>
<pre>void csipl_vmax_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum, by element, of two vectors.</p>
<pre>void csipl_vmaxmg_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum magnitude (absolute value), by element, of two vectors.</p>
<pre>void csipl_vcmaxmgsq_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum magnitude squared, by element, of two complex vectors.</p>
<pre>void csipl_vcmaxmgsq_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the maximum magnitude squared, by element, of two complex vectors.</p>
<pre>float csipl_vcmaxmgsqval_inter_f(void * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	<p>Returns the index and value of the maximum magnitude squared of the elements of a complex vector. The index is returned by reference as one of the arguments.</p>

Prototype	Description
<pre>float csipl_vcmaxmgsqval_split_f(float * A_re, float * A_im, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	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.
<pre>float csipl_vmaxmgval_f(float * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	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.
<pre>float csipl_vmaxval_f(float * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	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.
<pre>void csipl_vmin_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the minimum, by element, of two vectors.
<pre>void csipl_vminmg_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the minimum magnitude (absolute value), by element, of two vectors.
<pre>void csipl_vcminmgsq_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the minimum magnitude squared, by element, of two complex vectors.
<pre>void csipl_vcminmgsq_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, float * R, csipl_stride strideR, csipl_length n);</pre>	Computes the minimum magnitude squared, by element, of two complex vectors.

Prototype	Description
<pre>float csipl_vcminmgsqval_inter_f(void * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	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.
<pre>float csipl_vcminmgsqval_split_f(float * A_re, float * A_im, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	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.
<pre>float csipl_vminmgval_f(float * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	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.
<pre>float csipl_vminval_f(float * A, csipl_stride strideA, csipl_index * index, csipl_length n);</pre>	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.

4.7 Bitwise and Boolean Logical Operators

Prototype	Description
<pre>void csipl_vand_P(scalar_P * A, csipl_stride strideA, scalar_P * B, csipl_stride strideB, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the bitwise and, by element, of two vectors. The following instances are supported:</p> <pre>csipl_vand_i csipl_vand_si csipl_vand_bl</pre>
<pre>void csipl_mand_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Computes the bitwise and, by element, of two matrices. The following instances are supported:</p> <pre>csipl_mand_i csipl_mand_si csipl_mand_bl</pre>
<pre>void csipl_vnot_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes the bitwise not (one's complement), by element, of two vectors. The following instances are supported:</p> <pre>csipl_vnot_i csipl_vnot_si csipl_vnot_bl</pre>

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

4.8 Element Generation and Copy

Prototype	Description
<pre>void csipl_vcopy_P_P(scalar_P * A, csipl_stride strideA, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_vcopy_f_f csipl_vcopy_f_i csipl_vcopy_f_si csipl_vcopy_f_bl csipl_vcopy_i_f csipl_vcopy_i_i csipl_vcopy_i_si csipl_vcopy_i_vi csipl_vcopy_si_f csipl_vcopy_si_i csipl_vcopy_si_si csipl_vcopy_bl_f csipl_vcopy_bl_bl csipl_vcopy_vi_i csipl_vcopy_vi_vi csipl_vcopy_mi_mi</pre>
<pre>void csipl_cvcopy_inter_f_f(void * A, csipl_stride strideA, void * R, csipl_stride strideR, csipl_length n);</pre>	<p>Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.</p>
<pre>void csipl_cvcopy_split_f_f(float * A_re, float * A_im, csipl_stride strideA, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	<p>Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.</p>
<pre>void csipl_mcopy_P_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>csipl_mcopy_f_f csipl_mcopy_f_i csipl_mcopy_f_si csipl_mcopy_f_bl csipl_mcopy_i_f csipl_mcopy_i_i csipl_mcopy_i_si csipl_mcopy_si_f csipl_mcopy_si_i csipl_mcopy_si_si csipl_mcopy_bl_f csipl_mcopy_bl_bl</pre>

Prototype	Description
<pre>void csipl_cmcopy_inter_f_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.
<pre>void csipl_cmcopy_split_f_f(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.
<pre>void csipl_vfill_P(scalar_P a, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	Fill a vector with a constant value. The following instances are supported: <code>csipl_vfill_f</code> <code>csipl_vfill_i</code> <code>csipl_vfill_si</code>
<pre>void csipl_cvfill_inter_f(csipl_cscalar_f a, void * R, csipl_stride strideR, csipl_length n);</pre>	Fill a vector with a constant value.
<pre>void csipl_cvfill_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Fill a vector with a constant value.
<pre>void csipl_mfill_P(scalar_P a, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	Fill a matrix with a constant value. The following instances are supported: <code>csipl_mfill_f</code> <code>csipl_mfill_i</code> <code>csipl_mfill_si</code>
<pre>void csipl_cmfill_inter_f(csipl_cscalar_f a, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Fill a matrix with a constant value.
<pre>void csipl_cmfill_split_f(csipl_cscalar_f a_re, csipl_cscalar_f a_im, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Fill a matrix with a constant value.

Prototype	Description
<pre>void csipl_vramp_P(scalar_P alpha, scalar_P beta, scalar_P * R, csipl_stride strideR, csipl_length n);</pre>	<p>Computes a vector ramp by starting at an initial value and incrementing each successive element by the ramp step size.</p> <p>The following instances are supported:</p> <pre>csipl_vramp_f csipl_vramp_i csipl_vramp_si</pre>

4.9 Manipulation Operations

Prototype	Description
<pre>void csipl_vcplx_inter_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, void * R, csipl_stride strideR, csipl_length n);</pre>	Form a complex vector from two real vectors.
<pre>void csipl_vcplx_split_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, float * R_re, float * R_im, csipl_stride strideR, csipl_length n);</pre>	Form a complex vector from two real vectors.
<pre>void csipl_mcplx_f(float * A, int ldA, float * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Form a complex matrix from two real matrices.
<pre>void csipl_vgather_P(scalar_P * X, csipl_stride strideX, unsigned int * I, csipl_stride strideI, scalar_P * Y, csipl_stride strideY, csipl_length n);</pre>	<p>The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p> <p>The following instances are supported:</p> <pre>csipl_vgather_f csipl_vgather_i csipl_vgather_si</pre>

Prototype	Description
<pre>void csipl_cvgather_inter_f(void * X, csipl_stride strideX, unsigned int * I, csipl_stride strideI, void * Y, csipl_stride strideY, csipl_length n);</pre>	<p>The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p>
<pre>void csipl_cvgather_split_f(float * X_re, float * X_im, csipl_stride strideX, unsigned int * I_re, unsigned int * I_im, csipl_stride strideI, float * Y_re, float * Y_im, csipl_stride strideY, csipl_length n);</pre>	<p>The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p>
<pre>void csipl_mgather_P(scalar_P * X, int ldX, csipl_scalar_mi * I, csipl_stride strideI, scalar_P * Y, csipl_stride strideY, csipl_length n);</pre>	<p>The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p> <p>The following instances are supported:</p> <pre>csipl_mgather_f csipl_mgather_i csipl_mgather_si</pre>
<pre>void csipl_cmgather_inter_f(void * X, int ldX, csipl_scalar_mi * I, csipl_stride strideI, void * Y, csipl_stride strideY, csipl_length n);</pre>	<p>The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p>
<pre>void csipl_cmgather_split_f(float * X_re, float * X_im, int ldX, csipl_scalar_mi * I_re, csipl_scalar_mi * I_im, csipl_stride strideI, float * Y_re, float * Y_im, csipl_stride strideY, csipl_length n);</pre>	<p>The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p>
<pre>void csipl_vimag_inter_f(void * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	<p>Extract the imaginary part of a complex vector.</p>

Prototype	Description
<pre>void csipl_vimag_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Extract the imaginary part of a complex vector.
<pre>void csipl_mimag_f(void * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Extract the imaginary part of a complex matrix.
<pre>void csipl_vpolar_inter_f(void * A, csipl_stride strideA, float * R, csipl_stride strideR, float * P, csipl_stride strideP, csipl_length n);</pre>	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.
<pre>void csipl_vpolar_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R, csipl_stride strideR, float * P, csipl_stride strideP, csipl_length n);</pre>	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.
<pre>void csipl_mpolar_f(void * A, int ldA, float * R, int ldR, float * P, int ldP, csipl_length m, csipl_length n);</pre>	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.
<pre>void csipl_vreal_inter_f(void * A, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Extract the real part of a complex vector.
<pre>void csipl_vreal_split_f(float * A_re, float * A_im, csipl_stride strideA, float * R, csipl_stride strideR, csipl_length n);</pre>	Extract the real part of a complex vector.

Prototype	Description
<pre>void csipl_mreal_f(void * A, int ldA, float * R, int ldR, csipl_length m, csipl_length n);</pre>	Extract the real part of a complex matrix.
<pre>void csipl_vrect_inter_f(float * R, csipl_stride strideR, float * P, csipl_stride strideP, void * A, csipl_stride strideA, csipl_length n);</pre>	Convert a pair of real vectors from complex polar to complex rectangular form.
<pre>void csipl_vrect_split_f(float * R, csipl_stride strideR, float * P, csipl_stride strideP, float * A_re, float * A_im, csipl_stride strideA, csipl_length n);</pre>	Convert a pair of real vectors from complex polar to complex rectangular form.
<pre>void csipl_mrect_f(float * R, int ldR, float * P, int ldP, void * A, int ldA, csipl_length m, csipl_length n);</pre>	Convert a pair of real matrices from complex polar to complex rectangular form.
<pre>void csipl_vscatter_P(scalar_P * X, csipl_stride strideX, scalar_P * Y, csipl_stride strideY, unsigned int * I, csipl_stride strideI, csipl_length n);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.</p> <p>The following instances are supported:</p> <pre>csipl_vscatter_f csipl_vscatter_i csipl_vscatter_si</pre>
<pre>void csipl_cvscatter_inter_f(void * X, csipl_stride strideX, void * Y, csipl_stride strideY, unsigned int * I, csipl_stride strideI, csipl_length n);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.</p>

Prototype	Description
<pre>void csipl_cvscatter_split_f(float * X_re, float * X_im, csipl_stride strideX, float * Y_re, float * Y_im, csipl_stride strideY, unsigned int * I_re, unsigned int * I_im, csipl_stride strideI, csipl_length n);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.</p>
<pre>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);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.</p> <p>The following instances are supported:</p> <pre>csipl_mscatter_f csipl_mscatter_i csipl_mscatter_si</pre>
<pre>void csipl_cmscatter_inter_f(void * X, csipl_stride strideX, void * Y, int ldY, csipl_scalar_mi * I, csipl_stride strideI, csipl_length n);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.</p>
<pre>void csipl_cmscatter_split_f(float * X_re, float * X_im, csipl_stride strideX, float * Y_re, float * Y_im, int ldY, csipl_scalar_mi * I_re, csipl_scalar_mi * I_im, csipl_stride strideI, csipl_length n);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.</p>
<pre>void csipl_cvswap_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_length n);</pre>	<p>Swap elements between two vectors.</p>
<pre>void csipl_cvswap_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, csipl_length n);</pre>	<p>Swap elements between two vectors.</p>

Prototype	Description
<pre>void csipl_mswap_P(scalar_P * A, int ldA, scalar_P * B, int ldB, csipl_length m, csipl_length n);</pre>	<p>Swap elements between two matrices. The following instances are supported:</p> <pre>csipl_mswap_f csipl_mswap_i csipl_mswap_si</pre>
<pre>void csipl_cmswap_inter_f(void * A, int ldA, void * B, int ldB, csipl_length m, csipl_length n);</pre>	<p>Swap elements between two matrices.</p>
<pre>void csipl_cmswap_split_f(float * A_re, float * A_im, int ldA, float * B_re, float * B_im, int ldB, csipl_length m, csipl_length n);</pre>	<p>Swap elements between two matrices.</p>

Chapter 5. Signal Processing Functions

5.1 FFT Functions

Prototype	Description
<pre><code>csipl_fft_f * csipl_ccfftip_create_f(csipl_index length, float scale, csipl_fft_dir dir, unsigned int ntimes, csipl_alg_hint hint);</code></pre>	Create a 1D FFT object.
<pre><code>csipl_fft_f * csipl_ccfftop_create_f(csipl_index length, float scale, csipl_fft_dir dir, unsigned int ntimes, csipl_alg_hint hint);</code></pre>	Create a 1D FFT object.
<pre><code>csipl_fft_f * csipl_crfftop_create_f(csipl_index length, float scale, unsigned int ntimes, csipl_alg_hint hint);</code></pre>	Create a 1D FFT object.
<pre><code>csipl_fft_f * csipl_rcfftop_create_f(csipl_index length, float scale, unsigned int ntimes, csipl_alg_hint hint);</code></pre>	Create a 1D FFT object.
<pre><code>void csipl_ccfftip_inter_f(csipl_fft_f * plan, void * xy, csipl_stride stridexy);</code></pre>	Apply a complex-to-complex Fast Fourier Transform (FFT).
<pre><code>void csipl_ccfftip_split_f(csipl_fft_f * plan, float * xy_re, float * xy_im, csipl_stride stridexy);</code></pre>	Apply a complex-to-complex Fast Fourier Transform (FFT).
<pre><code>void csipl_ccfftop_inter_f(csipl_fft_f * plan, void * x, csipl_stride stridex, void * y, csipl_stride stridey);</code></pre>	Apply a complex-to-complex Fast Fourier Transform (FFT).
<pre><code>void csipl_ccfftop_split_f(csipl_fft_f * plan, float * x_re, float * x_im, csipl_stride stridex, float * y_re, float * y_im, csipl_stride stridey);</code></pre>	Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype	Description
<pre>void csipl_crffftop_inter_f(csipl_fft_f * plan, void * x, csipl_stride stridex, float * y, csipl_stride stridey);</pre>	Apply a complex-to-real Fast Fourier Transform (FFT).
<pre>void csipl_crffftop_split_f(csipl_fft_f * plan, float * x_re, float * x_im, csipl_stride stridex, float * y, csipl_stride stridey);</pre>	Apply a complex-to-real Fast Fourier Transform (FFT).
<pre>void csipl_rcffftop_inter_f(csipl_fft_f * plan, float * x, csipl_stride stridex, void * y, csipl_stride stridey);</pre>	Apply a real-to-complex Fast Fourier Transform (FFT).
<pre>void csipl_rcffftop_split_f(csipl_fft_f * plan, float * x, csipl_stride stridex, float * y_re, float * y_im, csipl_stride stridey);</pre>	Apply a real-to-complex Fast Fourier Transform (FFT).
<pre>int csipl_fft_destroy_f(csipl_fft_f * plan);</pre>	Destroy an FFT object.
<pre>void csipl_fft_getattr_f(csipl_fft_f * plan, csipl_fft_attr_f * attr);</pre>	Return the attributes of an FFT object.
<pre>csipl_fftm_f * csipl_ccfftmop_create_f(csipl_index rows, csipl_index cols, float scale, csipl_fft_dir dir, csipl_major major, unsigned int ntimes, csipl_alg_hint hint);</pre>	Create a 1D multiple FFT object.
<pre>csipl_fftm_f * csipl_ccfftmip_create_f(csipl_index rows, csipl_index cols, float scale, csipl_fft_dir dir, csipl_major major, unsigned int ntimes, csipl_alg_hint hint);</pre>	Create a 1D multiple FFT object.

Prototype	Description
<pre> <i>csipl_fftm_f</i> * csipl_crfftmop_create_f(csipl_index rows, csipl_index cols, float scale, csipl_major major, unsigned int ntimes, csipl_alg_hint hint); </pre>	Create a 1D multiple FFT object.
<pre> <i>csipl_fftm_f</i> * csipl_rcfftmop_create_f(csipl_index rows, csipl_index cols, float scale, csipl_major major, unsigned int ntimes, csipl_alg_hint hint); </pre>	Create a 1D multiple FFT object.
<pre> int csipl_fftm_destroy_f(csipl_fftm_f * plan); </pre>	Destroy an FFT object.
<pre> void csipl_fftm_getattr_f(csipl_fftm_f * plan, csipl_fftm_attr_f * attr); </pre>	Return the attributes of an FFT object.
<pre> void csipl_ccfftmip_inter_f(csipl_fftm_f * plan, void * XY, int ldXY); </pre>	Apply a multiple complex-to-complex Fast Fourier Transform (FFT).
<pre> void csipl_ccfftmip_split_f(csipl_fftm_f * plan, float * XY_re, float * XY_im, int ldXY); </pre>	Apply a multiple complex-to-complex Fast Fourier Transform (FFT).
<pre> void csipl_ccfftmop_inter_f(csipl_fftm_f * plan, void * X, int ldX, void * Y, int ldY); </pre>	Apply a multiple complex-to-complex Fast Fourier Transform (FFT).
<pre> void csipl_ccfftmop_split_f(csipl_fftm_f * plan, float * X_re, float * X_im, int ldX, float * Y_re, float * Y_im, int ldY); </pre>	Apply a multiple complex-to-complex Fast Fourier Transform (FFT).
<pre> void csipl_crfftmop_inter_f(csipl_fftm_f * plan, void * X, int ldX, float * Y, int ldY); </pre>	Apply a multiple complex-to-real Fast Fourier Transform (FFT).

Prototype	Description
<pre>void csipl_crfftmop_split_f(csipl_fftm_f * plan, float * X_re, float * X_im, int ldX, float * Y, int ldY);</pre>	Apply a multiple complex-to-real Fast Fourier Transform (FFT).
<pre>void csipl_rcfftmop_inter_f(csipl_fftm_f * plan, float * X, int ldX, void * Y, int ldY);</pre>	Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT).
<pre>void csipl_rcfftmop_split_f(csipl_fftm_f * plan, float * X, int ldX, float * Y_re, float * Y_im, int ldY);</pre>	Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT).
<pre>csipl_fft2d_f * csipl_ccfft2dop_create_f(csipl_index rows, csipl_index cols, float scale, csipl_fft_dir dir, unsigned int ntimes, csipl_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>csipl_fft2d_f * csipl_ccfft2dip_create_f(csipl_index rows, csipl_index cols, float scale, csipl_fft_dir dir, unsigned int ntimes, csipl_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>csipl_fft2d_f * csipl_crfft2dop_create_f(csipl_index rows, csipl_index cols, float scale, unsigned int ntimes, csipl_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>csipl_fft2d_f * csipl_rcfft2dop_create_f(csipl_index rows, csipl_index cols, float scale, unsigned int ntimes, csipl_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>void csipl_ccfft2dip_inter_f(csipl_fft2d_f * plan, void * XY, int ldXY);</pre>	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype	Description
<pre>void csipl_ccfft2dip_split_f(csipl_fft2d_f * plan, float * XY_re, float * XY_im, int ldXY);</pre>	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).
<pre>void csipl_ccfft2dop_inter_f(csipl_fft2d_f * plan, void * X, int ldX, void * Y, int ldY);</pre>	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).
<pre>void csipl_ccfft2dop_split_f(csipl_fft2d_f * plan, float * X_re, float * X_im, int ldX, float * Y_re, float * Y_im, int ldY);</pre>	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).
<pre>void csipl_crfft2dop_inter_f(csipl_fft2d_f * plan, void * X, int ldX, float * Y, int ldY);</pre>	Apply a complex-to-real 2D Fast Fourier Transform (FFT).
<pre>void csipl_crfft2dop_split_f(csipl_fft2d_f * plan, float * X_re, float * X_im, int ldX, float * Y, int ldY);</pre>	Apply a complex-to-real 2D Fast Fourier Transform (FFT).
<pre>void csipl_rcfft2dop_inter_f(csipl_fft2d_f * plan, float * X, int ldX, void * Y, int ldY);</pre>	Apply a real-to-complex 2D Fast Fourier Transform (FFT).
<pre>void csipl_rcfft2dop_split_f(csipl_fft2d_f * plan, float * X, int ldX, float * Y_re, float * Y_im, int ldY);</pre>	Apply a real-to-complex 2D Fast Fourier Transform (FFT).
<pre>int csipl_fft2d_destroy_f(csipl_fft2d_f * plan);</pre>	Destroy an FFT object.
<pre>void csipl_fft2d_getattr_f(csipl_fft2d_f * plan, csipl_fft2d_attr_f * attr);</pre>	Return the attributes of an FFT object.

5.2 Convolution/Correlation Functions

Prototype	Description
<pre> <i>csipl_conv1d_f</i> * <i>csipl_conv1d_create_f</i>(float * <i>kernel</i>, <i>csipl_stride</i> <i>stridekernel</i>, <i>csipl_symmetry</i> <i>symm</i>, unsigned int <i>N</i>, unsigned int <i>D</i>, <i>csipl_support_region</i> <i>support</i>, unsigned int <i>ntimes</i>, <i>csipl_alg_hint</i> <i>hint</i>, <i>csipl_length</i> <i>n</i>); </pre>	Create a decimated 1D convolution filter object.
<pre> int <i>csipl_conv1d_destroy_f</i>(<i>csipl_conv1d_f</i> * <i>plan</i>); </pre>	Destroy a 1D convolution object.
<pre> void <i>csipl_conv1d_getattr_f</i>(<i>csipl_conv1d_f</i> * <i>plan</i>, <i>csipl_conv1d_attr_f</i> * <i>attr</i>); </pre>	Returns the attributes for a 1D convolution object.
<pre> void <i>csipl_convolve1d_f</i>(<i>csipl_conv1d_f</i> * <i>plan</i>, float * <i>x</i>, <i>csipl_stride</i> <i>stridex</i>, float * <i>y</i>, <i>csipl_stride</i> <i>stridey</i>, <i>csipl_length</i> <i>n</i>); </pre>	Compute a decimated real one-dimensional (1D) convolution of two vectors.
<pre> <i>csipl_conv2d_f</i> * <i>csipl_conv2d_create_f</i>(float * <i>H</i>, int <i>ldH</i>, <i>csipl_symmetry</i> <i>symm</i>, unsigned int <i>P</i>, unsigned int <i>Q</i>, unsigned int <i>D</i>, <i>csipl_support_region</i> <i>support</i>, unsigned int <i>ntimes</i>, <i>csipl_alg_hint</i> <i>hint</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	Create a decimated 2D convolution filter object.
<pre> int <i>csipl_conv2d_destroy_f</i>(<i>csipl_conv2d_f</i> * <i>plan</i>); </pre>	Destroy a 2D convolution object.
<pre> void <i>csipl_conv2d_getattr_f</i>(<i>csipl_conv2d_f</i> * <i>plan</i>, <i>csipl_conv2d_attr_f</i> * <i>attr</i>); </pre>	Returns the attributes for a 2D convolution object.
<pre> void <i>csipl_convolve2d_f</i>(<i>csipl_conv2d_f</i> * <i>plan</i>, float * <i>x</i>, int <i>ldx</i>, float * <i>y</i>, int <i>ldy</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	Compute a decimated real two-dimensional (2D) convolution of two matrices.
<pre> <i>csipl_Dcorrid_P</i> * <i>csipl_Dcorrid_create_P</i>(unsigned int <i>M</i>, unsigned int <i>N</i>, <i>csipl_support_region</i> <i>support</i>, unsigned int <i>ntimes</i>, <i>csipl_alg_hint</i> <i>hint</i>); </pre>	Create a 1D correlation object. The following instances are supported: <i>csipl_corr1d_create_f</i> <i>csipl_ccorrid_create_f</i>

Prototype	Description
<pre>int csipl_Dcorr1d_destroy_P(csipl_Dcorr1d_P * plan);</pre>	<p>Destroy a 1D correlation object. The following instances are supported:</p> <p>csipl_corr1d_destroy_f csipl_ccorr1d_destroy_f</p>
<pre>void csipl_Dcorr1d_getattr_P(csipl_Dcorr1d_P * plan, csipl_Dcorr1d_attr_P * attr);</pre>	<p>Return the attributes for a 1D correlation object. The following instances are supported:</p> <p>csipl_corr1d_getattr_f csipl_ccorr1d_getattr_f</p>
<pre>void csipl_correlate1d_f(csipl_corr1d_f * plan, csipl_bias bias, float * ref, csipl_stride strideref, float * x, csipl_stride stridex, float * y, csipl_stride stridey, csipl_length n);</pre>	<p>Compute a real one-dimensional (1D) correlation of two vectors.</p>
<pre>void csipl_ccorrelate1d_inter_f(csipl_ccorr1d_f * plan, csipl_bias bias, void * ref, csipl_stride strideref, void * x, csipl_stride stridex, void * y, csipl_stride stridey, csipl_length n);</pre>	<p>Compute a real one-dimensional (1D) correlation of two vectors.</p>
<pre>void csipl_ccorrelate1d_split_f(csipl_ccorr1d_f * plan, csipl_bias bias, float * ref_re, float * ref_im, csipl_stride strideref, float * x_re, float * x_im, csipl_stride stridex, float * y_re, float * y_im, csipl_stride stridey, csipl_length n);</pre>	<p>Compute a real one-dimensional (1D) correlation of two vectors.</p>
<pre>csipl_Dcorr2d_P * csipl_Dcorr2d_create_P(unsigned int M, unsigned int N, unsigned int P, unsigned int Q, csipl_support_region support, unsigned int ntimes, csipl_alg_hint hint);</pre>	<p>Create a 2D correlation object. The following instances are supported:</p> <p>csipl_corr2d_create_f csipl_ccorr2d_create_f</p>
<pre>int csipl_Dcorr2d_destroy_P(csipl_Dcorr2d_P * plan);</pre>	<p>Destroy a 2D correlation object. The following instances are supported:</p> <p>csipl_corr2d_destroy_f csipl_ccorr2d_destroy_f</p>

Prototype	Description
<pre>void csipl_Dcorr2d_getattr_P(csipl_Dcorr2d_P * plan, csipl_Dcorr2d_attr_P * attr);</pre>	<p>Return the attributes for a 2D correlation object. The following instances are supported:</p> <p><code>csipl_corr2d_getattr_f</code> <code>csipl_ccorr2d_getattr_f</code></p>
<pre>void csipl_correlate2d_f(csipl_corr2d_f * plan, csipl_bias bias, float * ref, int ldref, float * x, int ldx, float * y, int ldy, csipl_length m, csipl_length n);</pre>	<p>Compute a two-dimensional (2D) correlation of two matrices.</p>
<pre>void csipl_ccorrelate2d_inter_f(csipl_ccorr2d_f * plan, csipl_bias bias, void * ref, int ldref, void * x, int ldx, void * y, int ldy, csipl_length m, csipl_length n);</pre>	<p>Compute a two-dimensional (2D) correlation of two matrices.</p>
<pre>void csipl_ccorrelate2d_split_f(csipl_ccorr2d_f * plan, csipl_bias bias, float * ref_re, float * ref_im, int ldref, float * x_re, float * x_im, int ldx, float * y_re, float * y_im, int ldy, csipl_length m, csipl_length n);</pre>	<p>Compute a two-dimensional (2D) correlation of two matrices.</p>

5.3 Window Functions

Prototype	Description
<pre>float * csipl_vcreate_blackman_f(unsigned int N, csipl_memory_hint hint);</pre>	<p>Create a vector with Blackman window weights.</p>
<pre>float * csipl_vcreate_cheby_f(unsigned int N, float ripple, csipl_memory_hint hint);</pre>	<p>Create a vector with Dolph-Chebyshev window weights.</p>

Prototype	Description
<pre>float * csipl_vcreate_hanning_f(unsigned int N, csipl_memory_hint hint);</pre>	Create a vector with Hanning window weights.
<pre>float * csipl_vcreate_kaiser_f(unsigned int N, float beta, csipl_memory_hint hint);</pre>	Create a vector with Kaiser window weights.

5.4 Filter Functions

Prototype	Description
<pre>csipl_fir_f * csipl_fir_create_f(float * kernel, csipl_stride stridekernel, csipl_symmetry symm, unsigned int N, unsigned int D, csipl_obj_state state, unsigned int ntimes, csipl_alg_hint hint, csipl_length n);</pre>	Create a decimated FIR filter object.
<pre>csipl_cfir_f * csipl_cfir_create_inter_f(void * kernel, csipl_stride stridekernel, csipl_symmetry symm, unsigned int N, unsigned int D, csipl_obj_state state, unsigned int ntimes, csipl_alg_hint hint, csipl_length n);</pre>	Create a decimated FIR filter object.
<pre>csipl_cfir_f * csipl_cfir_create_split_f(float * kernel_re, float * kernel_im, csipl_stride stridekernel, csipl_symmetry symm, unsigned int N, unsigned int D, csipl_obj_state state, unsigned int ntimes, csipl_alg_hint hint, csipl_length n);</pre>	Create a decimated FIR filter object.
<pre>int csipl_Dfir_destroy_P(csipl_Dfir_P * plan);</pre>	Destroy a FIR filter object. The following instances are supported: <code>csipl_fir_destroy_f</code> <code>csipl_cfir_destroy_f</code>
<pre>int csipl_firflt_f(csipl_fir_f * plan, float * x, csipl_stride stridex, float * y, csipl_stride stridey, csipl_length n);</pre>	FIR filter an input sequence and decimate the output.

Prototype	Description
<pre>int csipl_cfirflt_inter_f(csipl_cfir_f * plan, void * x, csipl_stride stridex, void * y, csipl_stride stridey, csipl_length n);</pre>	<p>FIR filter an input sequence and decimate the output.</p>
<pre>int csipl_cfirflt_split_f(csipl_cfir_f * plan, float * x_re, float * x_im, csipl_stride stridex, float * y_re, float * y_im, csipl_stride stridey, csipl_length n);</pre>	<p>FIR filter an input sequence and decimate the output.</p>
<pre>void csipl_Dfir_getattr_P(csipl_Dfir_P * plan, csipl_Dfir_attr_P * attr);</pre>	<p>Return the attributes of a FIR filter object. The following instances are supported:</p> <p><code>csipl_fir_getattr_f</code> <code>csipl_cfir_getattr_f</code></p>
<pre>void csipl_Dfir_reset_P(csipl_Dfir_P * fir);</pre>	<p>Reset the state of a decimated FIR filter object. The following instances are supported:</p> <p><code>csipl_fir_reset_f</code> <code>csipl_cfir_reset_f</code></p>

5.5 Miscellaneous Signal Processing Functions

Prototype	Description
<pre>void csipl_vhisto_f(float * A, csipl_stride strideA, float min, float max, csipl_hist_opt opt, float * R, csipl_stride strideR, csipl_length n);</pre>	Compute the histogram of a vector.

Chapter 6. Linear Algebra

6.1 Matrix and Vector Operations

Prototype	Description
<pre>void csipl_cmherm_inter_f(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	Complex Hermitian (conjugate transpose) of a matrix.
<pre>void csipl_cmherm_split_f(float * A_re, float * A_im, int ldA, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n);</pre>	Complex Hermitian (conjugate transpose) of a matrix.
<pre>csipl_cscalar_f csipl_cvjdot_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_length n);</pre>	Compute the conjugate inner (dot) product of two complex vectors.
<pre>csipl_cscalar_f csipl_cvjdot_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, csipl_length n);</pre>	Compute the conjugate inner (dot) product of two complex vectors.
<pre>void csipl_gemp_f(float alpha, float * A, int ldA, csipl_mat_op Aop, float * B, int ldB, csipl_mat_op Bop, float beta, float * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	Calculate the general product of two matrices and accumulate.

Prototype	Description
<pre>void csipl_cgemp_inter_f(csipl_cscalar_f alpha, void * A, int ldA, csipl_mat_op Aop, void * B, int ldB, csipl_mat_op Bop, csipl_cscalar_f beta, void * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	Calculate the general product of two matrices and accumulate.
<pre>void csipl_cgemp_split_f(csipl_cscalar_f alpha_re, csipl_cscalar_f alpha_im, float * A_re, float * A_im, int ldA, csipl_mat_op Aop, float * B_re, float * B_im, int ldB, csipl_mat_op Bop, csipl_cscalar_f beta_re, csipl_cscalar_f beta_im, float * R_re, float * R_im, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	Calculate the general product of two matrices and accumulate.
<pre>void csipl_gems_f(float alpha, float * A, int ldA, csipl_mat_op Aop, float beta, float * C, int ldC, csipl_length m, csipl_length n);</pre>	Calculate a general matrix sum.
<pre>void csipl_cgems_inter_f(csipl_cscalar_f alpha, void * A, int ldA, csipl_mat_op Aop, csipl_cscalar_f beta, void * C, int ldC, csipl_length m, csipl_length n);</pre>	Calculate a general matrix sum.

Prototype	Description
<pre>void csipl_cgems_split_f(csipl_cscalar_f alpha_re, csipl_cscalar_f alpha_im, float * A_re, float * A_im, int ldA, csipl_mat_op Aop, csipl_cscalar_f beta_re, csipl_cscalar_f beta_im, float * C_re, float * C_im, int ldC, csipl_length m, csipl_length n);</pre>	<p>Calculate a general matrix sum.</p>
<pre>void csipl_mprod_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product of two matrices. The following instances are supported:</p> <pre>csipl_mprod_f csipl_mprod_i csipl_mprod_si</pre>
<pre>void csipl_cmprod_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product of two matrices. The following instances are supported:</p> <pre>csipl_cmprod_inter_f csipl_cmprod_inter_i csipl_cmprod_inter_si</pre>
<pre>void csipl_cmprod_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B_re, scalar_P * B_im, int ldB, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product of two matrices. The following instances are supported:</p> <pre>csipl_cmprod_split_f csipl_cmprod_split_i csipl_cmprod_split_si</pre>
<pre>void csipl_cmprodh_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product a complex matrix and the Hermitian of a complex matrix. The following instances are supported:</p> <pre>csipl_cmprodh_inter_f csipl_cmprodh_inter_i csipl_cmprodh_inter_si</pre>

Prototype	Description
<pre>void csipl_cmprodh_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B_re, scalar_P * B_im, int ldB, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product a complex matrix and the Hermitian of a complex matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodh_split_f csipl_cmprodh_split_i csipl_cmprodh_split_si</pre>
<pre>void csipl_cmprodj_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product a complex matrix and the conjugate of a complex matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodj_inter_f csipl_cmprodj_inter_i csipl_cmprodj_inter_si</pre>
<pre>void csipl_cmprodj_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B_re, scalar_P * B_im, int ldB, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product a complex matrix and the conjugate of a complex matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodj_split_f csipl_cmprodj_split_i csipl_cmprodj_split_si</pre>
<pre>void csipl_mprodt_P(scalar_P * A, int ldA, scalar_P * B, int ldB, scalar_P * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product of a matrix and the transpose of a matrix.</p> <p>The following instances are supported:</p> <pre>csipl_mprodt_f csipl_mprodt_i csipl_mprodt_si</pre>
<pre>void csipl_cmprodt_inter_P(void * A, int ldA, void * B, int ldB, void * R, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product of a matrix and the transpose of a matrix.</p> <p>The following instances are supported:</p> <pre>csipl_cmprodt_inter_f csipl_cmprodt_inter_i csipl_cmprodt_inter_si</pre>

Prototype	Description
<pre>void csipl_cmprodt_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * B_re, scalar_P * B_im, int ldB, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n, csipl_length p);</pre>	<p>Calculate the product of a matrix and the transpose of a matrix. The following instances are supported:</p> <p><code>csipl_cmprodt_split_f</code> <code>csipl_cmprodt_split_i</code> <code>csipl_cmprodt_split_si</code></p>
<pre>void csipl_mvprod_P(scalar_P * A, int ldA, scalar_P * X, csipl_stride strideX, scalar_P * Y, csipl_stride strideY, csipl_length m, csipl_length n);</pre>	<p>Calculate a matrix–vector product. The following instances are supported:</p> <p><code>csipl_mvprod_f</code> <code>csipl_mvprod_i</code> <code>csipl_mvprod_si</code></p>
<pre>void csipl_cmvprod_inter_P(void * A, int ldA, void * X, csipl_stride strideX, void * Y, csipl_stride strideY, csipl_length m, csipl_length n);</pre>	<p>Calculate a matrix–vector product. The following instances are supported:</p> <p><code>csipl_cmvprod_inter_f</code> <code>csipl_cmvprod_inter_i</code> <code>csipl_cmvprod_inter_si</code></p>
<pre>void csipl_cmvprod_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * X_re, scalar_P * X_im, csipl_stride strideX, scalar_P * Y_re, scalar_P * Y_im, csipl_stride strideY, csipl_length m, csipl_length n);</pre>	<p>Calculate a matrix–vector product. The following instances are supported:</p> <p><code>csipl_cmvprod_split_f</code> <code>csipl_cmvprod_split_i</code> <code>csipl_cmvprod_split_si</code></p>
<pre>void csipl_mtrans_P(scalar_P * A, int ldA, scalar_P * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Transpose a matrix. The following instances are supported:</p> <p><code>csipl_mtrans_bl</code> <code>csipl_mtrans_f</code> <code>csipl_mtrans_i</code> <code>csipl_mtrans_si</code></p>
<pre>void csipl_cmtrans_inter_P(void * A, int ldA, void * R, int ldR, csipl_length m, csipl_length n);</pre>	<p>Transpose a matrix. The following instances are supported:</p> <p><code>csipl_cmtrans_inter_f</code> <code>csipl_cmtrans_inter_i</code> <code>csipl_cmtrans_inter_si</code></p>

Prototype	Description
<pre>void csipl_cmtrans_split_P(scalar_P * A_re, scalar_P * A_im, int ldA, scalar_P * R_re, scalar_P * R_im, int ldR, csipl_length m, csipl_length n);</pre>	<p>Transpose a matrix. The following instances are supported:</p> <pre>csipl_cmtrans_split_f csipl_cmtrans_split_i csipl_cmtrans_split_si</pre>
<pre>float csipl_vdot_f(float * A, csipl_stride strideA, float * B, csipl_stride strideB, csipl_length n);</pre>	<p>Compute the inner (dot) product of two vectors.</p>
<pre>csipl_cscalar_f csipl_cvdot_inter_f(void * A, csipl_stride strideA, void * B, csipl_stride strideB, csipl_length n);</pre>	<p>Compute the inner (dot) product of two vectors.</p>
<pre>csipl_cscalar_f csipl_cvdot_split_f(float * A_re, float * A_im, csipl_stride strideA, float * B_re, float * B_im, csipl_stride strideB, csipl_length n);</pre>	<p>Compute the inner (dot) product of two vectors.</p>
<pre>void csipl_vmprod_P(scalar_P * X, csipl_stride strideX, scalar_P * A, int ldA, scalar_P * Y, csipl_stride strideY, csipl_length m, csipl_length n);</pre>	<p>Calculate a vector–matrix product. The following instances are supported:</p> <pre>csipl_vmprod_f csipl_vmprod_i csipl_vmprod_si</pre>
<pre>void csipl_cvmprod_inter_P(void * X, csipl_stride strideX, void * A, int ldA, void * Y, csipl_stride strideY, csipl_length m, csipl_length n);</pre>	<p>Calculate a vector–matrix product. The following instances are supported:</p> <pre>csipl_cvmprod_inter_f csipl_cvmprod_inter_i csipl_cvmprod_inter_si</pre>

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

Prototype	Description
<pre>void csipl_minvlu_f(float * A, int ldA, signed int * V, csipl_stride strideV, float * R, int ldR, csipl_length n);</pre>	Invert a square matrix using LU decomposition.
<pre>void csipl_cminvlu_inter_f(void * A, int ldA, signed int * V, csipl_stride strideV, void * R, int ldR, csipl_length n);</pre>	Invert a square matrix using LU decomposition.
<pre>void csipl_cminvlu_split_f(float * A_re, float * A_im, int ldA, signed int * V, csipl_stride strideV, float * R_re, float * R_im, int ldR, csipl_length n);</pre>	Invert a square matrix using LU decomposition.

6.2 Special Linear System Solvers

Prototype	Description
<pre>int csipl_covsol_f(float * A, int ldA, float * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	Solve a covariance linear system problem.
<pre>int csipl_ccovsol_inter_f(void * A, int ldA, void * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	Solve a covariance linear system problem.
<pre>int csipl_ccovsol_split_f(float * A_re, float * A_im, int ldA, float * XB_re, float * XB_im, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	Solve a covariance linear system problem.
<pre>int csipl_llsqsol_f(float * A, int ldA, float * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	Solve a linear least squares problem.
<pre>int csipl_cllsqsol_inter_f(void * A, int ldA, void * XB, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	Solve a linear least squares problem.
<pre>int csipl_cllsqsol_split_f(float * A_re, float * A_im, int ldA, float * XB_re, float * XB_im, int ldXB, csipl_length m, csipl_length n, csipl_length p);</pre>	Solve a linear least squares problem.

Prototype	Description
<pre>int csipl_toepsol_f(float * T, csipl_stride strideT, float * B, csipl_stride strideB, float * W, csipl_stride strideW, float * X, csipl_stride strideX, csipl_length n);</pre>	Solve a real symmetric positive definite Toeplitz linear system.
<pre>int csipl_ctoepsol_inter_f(void * T, csipl_stride strideT, void * B, csipl_stride strideB, void * W, csipl_stride strideW, void * X, csipl_stride strideX, csipl_length n);</pre>	Solve a real symmetric positive definite Toeplitz linear system.
<pre>int csipl_ctoepsol_split_f(float * T_re, float * T_im, csipl_stride strideT, float * B_re, float * B_im, csipl_stride strideB, float * W_re, float * W_im, csipl_stride strideW, float * X_re, float * X_im, csipl_stride strideX, csipl_length n);</pre>	Solve a real symmetric positive definite Toeplitz linear system.

6.3 General Square Linear System Solver

Prototype	Description
<pre>int csipl_lud_f(csipl_clu_f * lud, float * A, int ldA, csipl_length n);</pre>	Compute an LU decomposition of a square matrix using partial pivoting.
<pre>int csipl_clud_inter_f(csipl_clu_f * lud, void * A, int ldA, csipl_length n);</pre>	Compute an LU decomposition of a square matrix using partial pivoting.
<pre>int csipl_clud_split_f(csipl_clu_f * lud, float * A_re, float * A_im, int ldA, csipl_length n);</pre>	Compute an LU decomposition of a square matrix using partial pivoting.

Prototype	Description
<pre> <i>csipl_Dlu_P</i> * <i>csipl_Dlud_create_P</i>(unsigned int <i>N</i>); </pre>	<p>Create an LU decomposition object. The following instances are supported:</p> <p><code><i>csipl_lud_create_f</i></code> <code><i>csipl_clud_create_f</i></code></p>
<pre> int <i>csipl_Dlud_destroy_P</i>(<i>csipl_Dlu_P</i> * <i>lud</i>); </pre>	<p>Destroy an LU decomposition object. The following instances are supported:</p> <p><code><i>csipl_lud_destroy_f</i></code> <code><i>csipl_clud_destroy_f</i></code></p>
<pre> void <i>csipl_Dlud_getattr_P</i>(<i>csipl_Dlu_P</i> * <i>lud</i>, <i>csipl_Dlu_attr_P</i> * <i>attr</i>); </pre>	<p>Returns the attributes of an LU decomposition object. The following instances are supported:</p> <p><code><i>csipl_lud_getattr_f</i></code> <code><i>csipl_clud_getattr_f</i></code></p>
<pre> int <i>csipl_lusol_f</i>(<i>csipl_clu_f</i> * <i>clud</i>, <i>csipl_mat_op</i> <i>opA</i>, float * <i>XB</i>, int <i>ldXB</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	Solve a square linear system.
<pre> int <i>csipl_clusol_inter_f</i>(<i>csipl_clu_f</i> * <i>clud</i>, <i>csipl_mat_op</i> <i>opA</i>, void * <i>XB</i>, int <i>ldXB</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	Solve a square linear system.
<pre> int <i>csipl_clusol_split_f</i>(<i>csipl_clu_f</i> * <i>clud_re</i>, <i>csipl_clu_f</i> * <i>clud_im</i>, <i>csipl_mat_op</i> <i>opA</i>, float * <i>XB_re</i>, float * <i>XB_im</i>, int <i>ldXB</i>, <i>csipl_length</i> <i>m</i>, <i>csipl_length</i> <i>n</i>); </pre>	Solve a square linear system.

6.4 Symmetric Positive Definite Linear System Solver

Prototype	Description
<pre> int <i>csipl_choold_f</i>(<i>csipl_cchol_f</i> * <i>choold</i>, float * <i>A</i>, int <i>ldA</i>, <i>csipl_length</i> <i>n</i>); </pre>	Compute a Cholesky decomposition of a symmetric positive definite matrix.
<pre> int <i>csipl_cchoold_inter_f</i>(<i>csipl_cchol_f</i> * <i>choold</i>, void * <i>A</i>, int <i>ldA</i>, <i>csipl_length</i> <i>n</i>); </pre>	Compute a Cholesky decomposition of a symmetric positive definite matrix.

Prototype	Description
<pre>int csipl_cchold_split_f(csipl_cchol_f * chold, float * A_re, float * A_im, int ldA, csipl_length n);</pre>	Compute a Cholesky decomposition of a symmetric positive definite matrix.
<pre>csipl_chol_f * csipl_chold_create_f(csipl_mat_uplo uplo, unsigned int n);</pre>	Creates a Cholesky decomposition object.
<pre>csipl_cchol_f * csipl_cchold_create_f(csipl_mat_uplo uplo, unsigned int n);</pre>	Creates a Cholesky decomposition object.
<pre>int csipl_Dchold_destroy_P(csipl_Dchol_P * chold);</pre>	Destroy a Cholesky decomposition object. The following instances are supported: <code>csipl_chold_destroy_f</code> <code>csipl_cchold_destroy_f</code>
<pre>void csipl_Dchold_getattr_P(csipl_Dchol_P * chold, csipl_Dchol_attr_P * attr);</pre>	Returns the attributes of a Cholesky decomposition object. The following instances are supported: <code>csipl_chold_getattr_f</code> <code>csipl_cchold_getattr_f</code>
<pre>int csipl_cholsol_f(csipl_cchol_f * chold, float * XB, int ldXB, csipl_length m, csipl_length n);</pre>	Solve a symmetric positive definite linear system.
<pre>int csipl_ccholsol_inter_f(csipl_cchol_f * chold, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	Solve a symmetric positive definite linear system.
<pre>int csipl_ccholsol_split_f(csipl_cchol_f * chold, float * XB_re, float * XB_im, int ldXB, csipl_length m, csipl_length n);</pre>	Solve a symmetric positive definite linear system.

6.5 Overdetermined Linear System Solver

Prototype	Description
<pre>int csipl_qrd_f(csipl_cqr_f * qrd, float * A, int ldA, csipl_length m, csipl_length n);</pre>	Compute a QR decomposition of a matrix .

Prototype	Description
<pre>int csipl_cqrd_inter_f(csipl_cqr_f * qrd, void * A, int ldA, csipl_length m, csipl_length n);</pre>	Compute a QR decomposition of a matrix .
<pre>int csipl_cqrd_split_f(csipl_cqr_f * qrd, float * A_re, float * A_im, int ldA, csipl_length m, csipl_length n);</pre>	Compute a QR decomposition of a matrix .
<pre>csipl_qr_f * csipl_qrd_create_f(unsigned int m, unsigned int n, csipl_qrd_qopt qopt);</pre>	Create a QR decomposition object.
<pre>csipl_cqr_f * csipl_cqrd_create_f(unsigned int m, unsigned int n, csipl_qrd_qopt qopt);</pre>	Create a QR decomposition object.
<pre>int csipl_Dqrd_destroy_P(csipl_Dqr_P * qrd);</pre>	Destroy a QR decomposition object. The following instances are supported: <code>csipl_qrd_destroy_f</code> <code>csipl_cqrd_destroy_f</code>
<pre>void csipl_Dqrd_getattr_P(csipl_Dqr_P * qrd, csipl_Dqr_attr_P * attr);</pre>	Returns the attributes of a QR decomposition object. The following instances are supported: <code>csipl_qrd_getattr_f</code> <code>csipl_cqrd_getattr_f</code>
<pre>int csipl_qrdprodq_f(csipl_qr_f * qrd, csipl_mat_op opQ, csipl_mat_side apQ, float * C, int ldC, csipl_length m, csipl_length n);</pre>	Multiply a matrix by the matrix Q from a QR decomposition.
<pre>int csipl_cqrdprodq_inter_f(csipl_cqr_f * qrd, csipl_mat_op opQ, csipl_mat_side apQ, void * C, int ldC, csipl_length m, csipl_length n);</pre>	Multiply a matrix by the matrix Q from a QR decomposition.

Prototype	Description
<pre>int csipl_cqrdprodq_split_f(csipl_cqr_f * qrd_re, csipl_cqr_f * qrd_im, csipl_mat_op opQ, csipl_mat_side apQ, float * C_re, float * C_im, int ldC, csipl_length m, csipl_length n);</pre>	Multiply a matrix by the matrix Q from a QR decomposition.
<pre>int csipl_qrdsolr_f(csipl_qr_f * qrd, csipl_mat_op OpR, float alpha, float * XB, int ldXB, csipl_length m, csipl_length n);</pre>	Solve linear system based on the matrix R , from QR decomposition of the matrix A .
<pre>int csipl_cqrdsolr_inter_f(csipl_cqr_f * qrd, csipl_mat_op OpR, csipl_cscalar_f alpha, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	Solve linear system based on the matrix R , from QR decomposition of the matrix A .
<pre>int csipl_cqrdsolr_split_f(csipl_cqr_f * qrd_re, csipl_cqr_f * qrd_im, csipl_mat_op OpR, csipl_cscalar_f alpha_re, csipl_cscalar_f alpha_im, float * XB_re, float * XB_im, int ldXB, csipl_length m, csipl_length n);</pre>	Solve linear system based on the matrix R , from QR decomposition of the matrix A .
<pre>int csipl_qrsol_f(csipl_qr_f * qrd, csipl_qrd_prob prob, float * XB, int ldXB, csipl_length m, csipl_length n);</pre>	Solve either a linear covariance or linear least squares problem.
<pre>int csipl_cqrsol_inter_f(csipl_cqr_f * qrd, csipl_qrd_prob prob, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	Solve either a linear covariance or linear least squares problem.

Prototype	Description
<pre>int csipl_cqrsol_split_f(csipl_cqr_f * qrd_re, csipl_cqr_f * qrd_im, csipl_qrd_prob prob, void * XB, int ldXB, csipl_length m, csipl_length n);</pre>	Solve either a linear covariance or linear least squares problem.