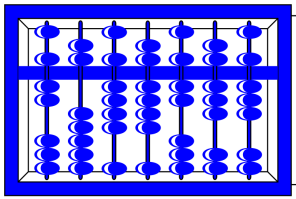

NASoftware Limited

Incorporating InfoSAR

# VSIPL
# List of Contents

**VSIPL/Content [3.20.14]**

**Release 3.20.14**
**February 2021**

# Chapter 1. Support Functions

## 1.1 Initialization and Finalization

| Prototype | Description |
|---|---|
| *int*<br>vsip_init(<br>void * ptr); | Provides initialization, allowing the library to allocate and set a global state, and prepare to support the use of VSIPL functionality by the user. |
| *int*<br>vsip_finalize(<br>void * ptr); | Provides cleanup and releases resources used by VSIPL (if the last of a nested series of calls), allowing the library to guarantee that any resources allocated by vsip_init are no longer in use after the call is complete. |
| *void*<br>Thread_SetParams(<br>vsip_length num_threads,<br>vsip_length max_num_running); | Allows the library to allocate a number of threads. |

## 1.2 Array and Block Functions

| Prototype | Description |
|---|---|
| *int*<br>vsip_Dblockadmit_P(<br>vsip_Dblock_P * block,<br>vsip_scalar_bl update); | Admit a data block for VSIPL operations.<br>The following instances are supported:<br>vsip_blockadmit_f<br>vsip_blockadmit_i<br>vsip_cblockadmit_f<br>vsip_blockadmit_bl<br>vsip_blockadmit_vi<br>vsip_blockadmit_mi |
| *vsip_block_P **<br>vsip_blockbind_P(<br>vsip_scalar_P * user_data,<br>vsip_length num_items,<br>vsip_memory_hint hint); | Create and bind a VSIPL block to a user-allocated data array.<br>The following instances are supported:<br>vsip_blockbind_f<br>vsip_blockbind_i<br>vsip_blockbind_bl<br>vsip_blockbind_vi<br>vsip_blockbind_mi |
| *vsip_cblock_f **<br>vsip_cblockbind_f(<br>vsip_scalar_f * user_data1,<br>vsip_scalar_f * user_data2,<br>vsip_length num_items,<br>vsip_memory_hint hint); | Create and bind a VSIPL complex block to a user-allocated data array. |

| Prototype | Description |
|---|---|
| *vsip_Dblock_P* *<br>vsip_Dblockcreate_P(<br>vsip_length num_items,<br>vsip_memory_hint hint); | Creates a VSIPL block and binds a (VSIPL-allocated) data array to it.<br>The following instances are supported:<br>vsip_blockcreate_f<br>vsip_blockcreate_i<br>vsip_cblockcreate_f<br>vsip_blockcreate_bl<br>vsip_blockcreate_vi<br>vsip_blockcreate_mi |
| *void*<br>vsip_Dblockdestroy_P(<br>vsip_Dblock_P * block); | Destroy a VSIPL block object and any memory allocated for it by VSIPL.<br>The following instances are supported:<br>vsip_blockdestroy_f<br>vsip_blockdestroy_i<br>vsip_cblockdestroy_f<br>vsip_blockdestroy_bl<br>vsip_blockdestroy_vi<br>vsip_blockdestroy_mi |
| *vsip_scalar_P* *<br>vsip_blockfind_P(<br>const vsip_block_P * block); | Find the pointer to the data bound to a VSIPL released block object.<br>The following instances are supported:<br>vsip_blockfind_f<br>vsip_blockfind_i<br>vsip_blockfind_bl<br>vsip_blockfind_vi<br>vsip_blockfind_mi |
| *void*<br>vsip_cblockfind_f(<br>const vsip_cblock_f * block,<br>vsip_scalar_f ** user_data1,<br>vsip_scalar_f ** user_data2); | Find the pointer(s) to the data bound to a VSIPL released complex block object. |
| *vsip_scalar_P* *<br>vsip_blockrebind_P(<br>vsip_block_P * block,<br>vsip_scalar_P * new_data); | Rebind a VSIPL block to user-specified data.<br>The following instances are supported:<br>vsip_blockrebind_f<br>vsip_blockrebind_i<br>vsip_blockrebind_bl<br>vsip_blockrebind_vi<br>vsip_blockrebind_mi |
| *void*<br>vsip_cblockrebind_f(<br>vsip_cblock_f * block,<br>vsip_scalar_f * new_data1,<br>vsip_scalar_f * new_data2,<br>vsip_scalar_f ** old_data1,<br>vsip_scalar_f ** old_data2); | Rebind a VSIPL complex block to user-specified data. |
| *vsip_scalar_P* *<br>vsip_blockrelease_P(<br>vsip_block_P * block,<br>vsip_scalar_bl update); | Release a VSIPL block for direct user access.<br>The following instances are supported:<br>vsip_blockrelease_f<br>vsip_blockrelease_i<br>vsip_blockrelease_bl<br>vsip_blockrelease_vi<br>vsip_blockrelease_mi |

| Prototype | Description |
|---|---|
| *void*<br>vsip_cblockrelease_f(<br>vsip_cblock_f * block,<br>vsip_scalar_bl update,<br>vsip_scalar_f ** user_data1,<br>vsip_scalar_f ** user_data2); | Release a complex block from VSIPL for direct user access. |
| *vsip_cmplx_mem*<br>vsip_cstorage(<br>void); | Returns the preferred complex storage format for the system. |

## 1.3 Vector View Functions

| Prototype | Description |
|---|---|
| *void*<br>vsip_Dvalldestroy_P(<br>vsip_Dvview_P * vector); | Destroy a vector, its associated block, and any VSIPL data array bound to the block.<br>The following instances are supported:<br>vsip_valldestroy_f<br>vsip_valldestroy_i<br>vsip_cvalldestroy_f<br>vsip_valldestroy_bl<br>vsip_valldestroy_vi<br>vsip_valldestroy_mi |
| *vsip_Dvview_P **<br>vsip_Dvbind_P(<br>vsip_Dblock_P * block,<br>vsip_offset offset,<br>vsip_stride stride,<br>vsip_length length); | Create a vector view object and bind it to a block object.<br>The following instances are supported:<br>vsip_vbind_f<br>vsip_vbind_i<br>vsip_cvbind_f<br>vsip_vbind_bl<br>vsip_vbind_vi<br>vsip_vbind_mi |
| *vsip_Dvview_P **<br>vsip_Dvcloneview_P(<br>const vsip_Dvview_P * vector); | Create a clone of a vector view.<br>The following instances are supported:<br>vsip_vcloneview_f<br>vsip_vcloneview_i<br>vsip_cvcloneview_f<br>vsip_vcloneview_bl<br>vsip_vcloneview_vi<br>vsip_vcloneview_mi |
| *vsip_Dvview_P **<br>vsip_Dvcreate_P(<br>vsip_length length,<br>vsip_memory_hint hint); | Creates a block object and a vector view object of the block.<br>The following instances are supported:<br>vsip_vcreate_f<br>vsip_vcreate_i<br>vsip_cvcreate_f<br>vsip_vcreate_bl<br>vsip_vcreate_vi<br>vsip_vcreate_mi |

| Prototype | Description |
|---|---|
| *vsip_Dblock_P* *<br>vsip_Dvdestroy_P(<br>vsip_Dvview_P * vector); | Destroy a vector view object and return a pointer to the associated block object.<br>The following instances are supported:<br>vsip_vdestroy_f<br>vsip_vdestroy_i<br>vsip_cvdestroy_f<br>vsip_vdestroy_bl<br>vsip_vdestroy_vi<br>vsip_vdestroy_mi |
| *vsip_Dscalar_P*<br>vsip_Dvget_P(<br>const vsip_Dvview_P * vector,<br>vsip_index j); | Get the value of a specified element of a vector view object.<br>The following instances are supported:<br>vsip_vget_f<br>vsip_vget_i<br>vsip_cvget_f<br>vsip_vget_bl<br>vsip_vget_vi<br>vsip_vget_mi |
| *void*<br>vsip_Dvgetattrib_P(<br>const vsip_Dvview_P * vector,<br>vsip_Dvattr_P * attr); | Return the attributes of a vector view object.<br>The following instances are supported:<br>vsip_vgetattrib_f<br>vsip_vgetattrib_i<br>vsip_cvgetattrib_f<br>vsip_vgetattrib_bl<br>vsip_vgetattrib_vi<br>vsip_vgetattrib_mi |
| *vsip_Dblock_P* *<br>vsip_Dvgetblock_P(<br>const vsip_Dvview_P * vector); | Get the block attribute of a vector view object.<br>The following instances are supported:<br>vsip_vgetblock_f<br>vsip_vgetblock_i<br>vsip_cvgetblock_f<br>vsip_vgetblock_bl<br>vsip_vgetblock_vi<br>vsip_vgetblock_mi |
| *vsip_length*<br>vsip_Dvgetlength_P(<br>const vsip_Dvview_P * vector); | Get the length attribute of a vector view object.<br>The following instances are supported:<br>vsip_vgetlength_f<br>vsip_vgetlength_i<br>vsip_cvgetlength_f<br>vsip_vgetlength_bl<br>vsip_vgetlength_vi<br>vsip_vgetlength_mi |

| Prototype | Description |
|---|---|
| *vsip_offset*<br>vsip_Dvgetoffset_P(<br>const vsip_Dvview_P * vector); | Get the offset attribute of a vector view object.<br>The following instances are supported:<br>vsip_vgetoffset_f<br>vsip_vgetoffset_i<br>vsip_cvgetoffset_f<br>vsip_vgetoffset_bl<br>vsip_vgetoffset_vi<br>vsip_vgetoffset_mi |
| *vsip_stride*<br>vsip_Dvgetstride_P(<br>const vsip_Dvview_P * vector); | Get the stride attribute of a vector view object.<br>The following instances are supported:<br>vsip_vgetstride_f<br>vsip_vgetstride_i<br>vsip_cvgetstride_f<br>vsip_vgetstride_bl<br>vsip_vgetstride_vi<br>vsip_vgetstride_mi |
| *vsip_vview_f* *<br>vsip_vimagview_f(<br>const vsip_cvview_f *<br>complex_vector); | Create a vector view object of the imaginary part of a complex vector from a complex vector view object. |
| *void*<br>vsip_Dvput_P(<br>vsip_Dvview_P * vector,<br>vsip_index j,<br>vsip_Dscalar_P value); | Set the value of a specified element of a vector view object.<br>The following instances are supported:<br>vsip_vput_f<br>vsip_vput_i<br>vsip_cvput_f<br>vsip_vput_bl<br>vsip_vput_vi<br>vsip_vput_mi |
| *vsip_Dvview_P* *<br>vsip_Dvputattrib_P(<br>vsip_Dvview_P * ve3tor,<br>const vsip_Dvattr_P * attr); | Set the attributes of a vector view object.<br>The following instances are supported:<br>vsip_vputattrib_f<br>vsip_vputattrib_i<br>vsip_cvputattrib_f<br>vsip_vputattrib_bl<br>vsip_vputattrib_vi<br>vsip_vputattrib_mi |
| *vsip_Dvview_P* *<br>vsip_Dvputlength_P(<br>vsip_Dvview_P * vector,<br>vsip_length length); | Set the length attribute of a vector view object.<br>The following instances are supported:<br>vsip_vputlength_f<br>vsip_vputlength_i<br>vsip_cvputlength_f<br>vsip_vputlength_bl<br>vsip_vputlength_vi<br>vsip_vputlength_mi |

| Prototype | Description |
|---|---|
| *vsip_Dvview_P* * vsip_Dvputoffset_P( vsip_Dvview_P * vector, vsip_offset offset); | Set the offset attribute of a vector view object.<br>The following instances are supported:<br>vsip_vputoffset_f<br>vsip_vputoffset_i<br>vsip_cvputoffset_f<br>vsip_vputoffset_bl<br>vsip_vputoffset_vi<br>vsip_vputoffset_mi |
| *vsip_Dvview_P* * vsip_Dvputstride_P( vsip_Dvview_P * vector, vsip_stride stride); | Set the stride attribute of a vector view object.<br>The following instances are supported:<br>vsip_vputstride_f<br>vsip_vputstride_i<br>vsip_cvputstride_f<br>vsip_vputstride_bl<br>vsip_vputstride_vi<br>vsip_vputstride_mi |
| *vsip_vview_f* * vsip_vrealview_f( const vsip_cvview_f * complex_vector); | Create a vector view object of the real part of a complex vector from a complex vector view object. |
| *vsip_Dvview_P* * vsip_Dvsubview_P( const vsip_Dvview_P * vector, vsip_index j, vsip_length n); | Create a vector view object that is a subview of a vector view object.<br>The following instances are supported:<br>vsip_vsubview_f<br>vsip_vsubview_i<br>vsip_cvsubview_f<br>vsip_vsubview_bl<br>vsip_vsubview_vi<br>vsip_vsubview_mi |

## 1.4 Matrix View Functions

| Prototype | Description |
|---|---|
| *void* vsip_Dmalldestroy_P( vsip_Dmview_P * matrix); | Destroy a matrix, its associated block, and any VSIPL data array bound to the block.<br>The following instances are supported:<br>vsip_malldestroy_f<br>vsip_cmalldestroy_f |
| *vsip_Dmview_P* * vsip_Dmbind_P( vsip_Dblock_P * block, vsip_offset offset, vsip_stride col_stride, vsip_length col_length, vsip_stride row_stride, vsip_length row_length); | Create a matrix view object and bind it to a block object.<br>The following instances are supported:<br>vsip_mbind_f<br>vsip_cmbind_f |
| *vsip_Dmview_P* * vsip_Dmcloneview_P( const vsip_Dmview_P * matrix); | Create a clone of a matrix view.<br>The following instances are supported:<br>vsip_mcloneview_f<br>vsip_cmcloneview_f |

| Prototype | Description |
|---|---|
| *vsip_Dvview_P* *<br>vsip_Dmcolview_P(<br>const vsip_Dmview_P * matrix,<br>vsip_index j); | Create a vector view object of a specified column of the source matrix view object.<br>The following instances are supported:<br>vsip_mcolview_f<br>vsip_cmcolview_f |
| *vsip_Dmview_P* *<br>vsip_Dmcreate_P(<br>vsip_length col_length,<br>vsip_length row_length,<br>vsip_major rc,<br>vsip_memory_hint hint); | Creates a block object and matrix view object of the block.<br>The following instances are supported:<br>vsip_mcreate_f<br>vsip_cmcreate_f |
| *vsip_Dblock_P* *<br>vsip_Dmdestroy_P(<br>vsip_Dmview_P * matrix); | Destroy a matrix view object and returns a pointer to the associated block object.<br>The following instances are supported:<br>vsip_mdestroy_f<br>vsip_cmdestroy_f |
| *vsip_Dvview_P* *<br>vsip_Dmdiagview_P(<br>const vsip_Dmview_P * matrix,<br>vsip_stride index); | Create a vector view object of a matrix diagonal of a matrix view object.<br>The following instances are supported:<br>vsip_mdiagview_f<br>vsip_cmdiagview_f |
| *vsip_Dscalar_P*<br>vsip_Dmget_P(<br>const vsip_Dmview_P * matrix,<br>vsip_index i,<br>vsip_index j); | Get the value of a specified element of a matrix view object.<br>The following instances are supported:<br>vsip_mget_f<br>vsip_cmget_f |
| *void*<br>vsip_Dmgetattrib_P(<br>const vsip_Dmview_P * matrix,<br>vsip_Dmattr_P * attr); | Get the attributes of a matrix view object.<br>The following instances are supported:<br>vsip_mgetattrib_f<br>vsip_cmgetattrib_f |
| *vsip_Dblock_P* *<br>vsip_Dmgetblock_P(<br>const vsip_Dmview_P * matrix); | Get the block attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mgetblock_f<br>vsip_cmgetblock_f |
| *vsip_length*<br>vsip_Dmgetcollength_P(<br>const vsip_Dmview_P * matrix); | Get the column length attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mgetcollength_f<br>vsip_cmgetcollength_f |
| *vsip_stride*<br>vsip_Dmgetcolstride_P(<br>const vsip_Dmview_P * matrix); | Get the column stride attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mgetcolstride_f<br>vsip_cmgetcolstride_f |

| Prototype | Description |
|---|---|
| *vsip_offset*<br>vsip_Dmgetoffset_P(<br>const vsip_Dmview_P * matrix); | Get the offset attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mgetoffset_f<br>vsip_cmgetoffset_f |
| *vsip_length*<br>vsip_Dmgetrowlength_P(<br>const vsip_Dmview_P * matrix); | Get the row length attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mgetrowlength_f<br>vsip_cmgetrowlength_f |
| *vsip_stride*<br>vsip_Dmgetrowstride_P(<br>const vsip_Dmview_P * matrix); | Get the row stride attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mgetrowstride_f<br>vsip_cmgetrowstride_f |
| *vsip_mview_f **<br>vsip_mimagview_f(<br>const vsip_cmview_f *<br>complex_matrix); | Create a matrix view object of the imaginary part of complex matrix from a complex matrix view object. |
| *void*<br>vsip_Dmput_P(<br>const vsip_Dmview_P * matrix,<br>vsip_index i,<br>vsip_index j,<br>vsip_Dscalar_P value); | Set the value of a specified element of a matrix view object.<br>The following instances are supported:<br>vsip_mput_f<br>vsip_cmput_f |
| *vsip_Dmview_P **<br>vsip_Dmputattrib_P(<br>vsip_Dmview_P * matrix,<br>const vsip_Dmattr_P * attr); | Set the attributes of a matrix view object.<br>The following instances are supported:<br>vsip_mputattrib_f<br>vsip_cmputattrib_f |
| *vsip_Dmview_P **<br>vsip_Dmputcollength_P(<br>vsip_Dmview_P * matrix,<br>vsip_length n2); | Set the column length attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mputcollength_f<br>vsip_cmputcollength_f |
| *vsip_Dmview_P **<br>vsip_Dmputcolstride_P(<br>vsip_Dmview_P * matrix,<br>vsip_stride s2); | Set the column stride attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mputcolstride_f<br>vsip_cmputcolstride_f |
| *vsip_Dmview_P **<br>vsip_Dmputoffset_P(<br>vsip_Dmview_P * matrix,<br>vsip_offset offset); | Set the offset attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mputoffset_f<br>vsip_cmputoffset_f |
| *vsip_Dmview_P **<br>vsip_Dmputrowlength_P(<br>vsip_Dmview_P * matrix,<br>vsip_length n1); | Set the row length attribute of a matrix view object.<br>The following instances are supported:<br>vsip_mputrowlength_f<br>vsip_cmputrowlength_f |

| Prototype | Description |
|---|---|
| `vsip_Dmview_P *`<br>`vsip_Dmputrowstride_P(`<br>`vsip_Dmview_P * matrix,`<br>`vsip_stride s1);` | Set the row stride attribute of a matrix view object.<br>The following instances are supported:<br>`vsip_mputrowstride_f`<br>`vsip_cmputrowstride_f` |
| `vsip_mview_f *`<br>`vsip_mrealview_f(`<br>`const vsip_cmview_f *`<br>`complex_matrix);` | Create a matrix view object of the real part of complex matrix from a complex matrix view object. |
| `vsip_Dvview_P *`<br>`vsip_Dmrowview_P(`<br>`const vsip_Dmview_P * matrix,`<br>`vsip_index i);` | Create a vector view object of a specified row of the source matrix view object.<br>The following instances are supported:<br>`vsip_mrowview_f`<br>`vsip_cmrowview_f` |
| `vsip_Dmview_P *`<br>`vsip_Dmsubview_P(`<br>`const vsip_Dmview_P * matrix,`<br>`vsip_index i,`<br>`vsip_index j,`<br>`vsip_length m,`<br>`vsip_length n);` | Create a matrix view object that is a subview of matrix view object.<br>The following instances are supported:<br>`vsip_msubview_f`<br>`vsip_cmsubview_f` |
| `vsip_Dmview_P *`<br>`vsip_Dmtransview_P(`<br>`const vsip_Dmview_P * matrix);` | Create a matrix view object that is the transpose of a matrix view object.<br>The following instances are supported:<br>`vsip_mtransview_f`<br>`vsip_cmtransview_f` |

# Chapter 2. Scalar Functions

## 2.1 Complex Scalar Functions

| Prototype | Description |
|---|---|
| `vsip_scalar_f`<br>`vsip_arg_f(`<br>`vsip_cscalar_f x);` | Returns the argument in radians $[-\pi, \pi]$ of a complex scalar. |
| `void`<br>`vsip_CADD_f(`<br>`vsip_cscalar_f x,`<br>`vsip_cscalar_f y,`<br>`vsip_cscalar_f * z);` | Computes the complex sum of two scalars. |
| `vsip_cscalar_f`<br>`vsip_cadd_f(`<br>`vsip_cscalar_f x,`<br>`vsip_cscalar_f y);` | Computes the complex sum of two scalars. |
| `void`<br>`vsip_RCADD_f(`<br>`vsip_scalar_f x,`<br>`vsip_cscalar_f y,`<br>`vsip_cscalar_f * z);` | Computes the complex sum of two scalars. |
| `vsip_cscalar_f`<br>`vsip_rcadd_f(`<br>`vsip_scalar_f x,`<br>`vsip_cscalar_f y);` | Computes the complex sum of two scalars. |
| `void`<br>`vsip_CDIV_f(`<br>`vsip_cscalar_f x,`<br>`vsip_cscalar_f y,`<br>`vsip_cscalar_f * z);` | Computes the complex quotient of two scalars. |
| `vsip_cscalar_f`<br>`vsip_cdiv_f(`<br>`vsip_cscalar_f x,`<br>`vsip_cscalar_f y);` | Computes the complex quotient of two scalars. |
| `void`<br>`vsip_CRDIV_f(`<br>`vsip_cscalar_f x,`<br>`vsip_scalar_f y,`<br>`vsip_cscalar_f * z);` | Computes the complex quotient of two scalars. |
| `vsip_cscalar_f`<br>`vsip_crdiv_f(`<br>`vsip_cscalar_f x,`<br>`vsip_scalar_f y);` | Computes the complex quotient of two scalars. |
| `void`<br>`vsip_CEXP_f(`<br>`vsip_cscalar_f x,`<br>`vsip_cscalar_f * y);` | Computes the exponential of a scalar. |
| `vsip_cscalar_f`<br>`vsip_cexp_f(`<br>`const vsip_cscalar_f A);` | Computes the exponential of a scalar. |
| `void`<br>`vsip_CJMUL_f(`<br>`vsip_cscalar_f x,`<br>`vsip_cscalar_f y,`<br>`vsip_cscalar_f * z);` | Computes the product a complex scalar with the conjugate of a second complex scalar. |

| Prototype | Description |
|---|---|
| `vsip_cscalar_f vsip_cjmul_f( vsip_cscalar_f x, vsip_cscalar_f y);` | Computes the product a complex scalar with the conjugate of a second complex scalar. |
| `vsip_cscalar_f vsip_cmag_f( const vsip_cscalar_f A);` | Computes the magnitude (absolute value) of a scalar. |
| `vsip_scalar_f vsip_cmagsq_f( vsip_cscalar_f x);` | Computes the magnitude squared of a complex scalar. |
| `void vsip_CMPLX_f( vsip_scalar_f a, vsip_scalar_f b, vsip_cscalar_f * r);` | Form a complex scalar from two real scalars. |
| `vsip_cscalar_f vsip_cmplx_f( vsip_scalar_f re, vsip_scalar_f im);` | Form a complex scalar from two real scalars. |
| `void vsip_CMUL_f( vsip_cscalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);` | Computes the complex product of two scalars. |
| `vsip_cscalar_f vsip_cmul_f( vsip_cscalar_f x, vsip_cscalar_f y);` | Computes the complex product of two scalars. |
| `void vsip_RCMUL_f( vsip_scalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);` | Computes the complex product of two scalars. |
| `vsip_cscalar_f vsip_rcmul_f( vsip_scalar_f x, vsip_cscalar_f y);` | Computes the complex product of two scalars. |
| `void vsip_CNEG_f( vsip_cscalar_f x, vsip_cscalar_f * y);` | Computes the negation of a complex scalar. |
| `vsip_cscalar_f vsip_cneg_f( vsip_cscalar_f x);` | Computes the negation of a complex scalar. |
| `void vsip_CONJ_f( vsip_cscalar_f x, vsip_cscalar_f * y);` | Computes the complex conjugate of a scalar. |
| `vsip_cscalar_f vsip_conj_f( vsip_cscalar_f x);` | Computes the complex conjugate of a scalar. |
| `void vsip_CRECIP_f( vsip_cscalar_f x, vsip_cscalar_f * y);` | Computes the reciprocal of a complex scalar. |
| `vsip_cscalar_f vsip_crecip_f( vsip_cscalar_f x);` | Computes the reciprocal of a complex scalar. |
| `void vsip_CSQRT_f( vsip_cscalar_f x, vsip_cscalar_f * y);` | Computes the square root a complex scalar. |

| Prototype | Description |
|---|---|
| `vsip_cscalar_f vsip_csqrt_f( vsip_cscalar_f x);` | Computes the square root a complex scalar. |
| `void vsip_CSUB_f( vsip_cscalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);` | Computes the complex difference of two scalars. |
| `vsip_cscalar_f vsip_csub_f( vsip_cscalar_f x, vsip_cscalar_f y);` | Computes the complex difference of two scalars. |
| `void vsip_RCSUB_f( vsip_scalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);` | Computes the complex difference of two scalars. |
| `vsip_cscalar_f vsip_rcsub_f( vsip_scalar_f x, vsip_cscalar_f y);` | Computes the complex difference of two scalars. |
| `void vsip_CRSUB_f( vsip_cscalar_f x, vsip_scalar_f y, vsip_cscalar_f * z);` | Computes the complex difference of two scalars. |
| `vsip_cscalar_f vsip_crsub_f( vsip_cscalar_f x, vsip_scalar_f y);` | Computes the complex difference of two scalars. |
| `vsip_scalar_f vsip_imag_f( vsip_cscalar_f x);` | Extract the imaginary part of a complex scalar. |
| `void vsip_polar_f( vsip_cscalar_f a, vsip_scalar_f * r, vsip_scalar_f * t);` | Convert a complex scalar from rectangular to polar form. The polar data consists of a real scalar containing the radius and a corresponding real scalar containing the argument (angle) of the complex scalar. |
| `vsip_scalar_f vsip_real_f( vsip_cscalar_f x);` | Extract the real part of a complex scalar. |
| `vsip_cscalar_f vsip_rect_f( vsip_scalar_f r, vsip_scalar_f t);` | Convert a pair of real scalars from complex polar to complex rectangular form. |

## 2.2 Index Scalar Functions

| Prototype | Description |
|---|---|
| *void*<br>vsip_MATINDEX(<br>vsip_index r,<br>vsip_index c,<br>vsip_scalar_mi * mi); | Form a matrix index from two vector indices. |
| *vsip_scalar_mi*<br>vsip_matindex(<br>vsip_index r,<br>vsip_index c); | Form a matrix index from two vector indices. |
| *vsip_index*<br>vsip_mcolindex(<br>vsip_scalar_mi mi); | Returns the column vector index from a matrix index. |
| *vsip_index*<br>vsip_mrowindex(<br>vsip_scalar_mi mi); | Returns the row vector index from a matrix index. |

# Chapter 3. Random Number Generation

## 3.1 Random Number Functions

| Prototype | Description |
|---|---|
| `vsip_randstate *`<br>`vsip_randcreate(`<br>`const vsip_index seed,`<br>`const vsip_index numprocs,`<br>`const vsip_index id,`<br>`const vsip_rng portable);` | Create a random number generator state object. |
| `int`<br>`vsip_randdestroy(`<br>`vsip_randstate * rand);` | Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure. |
| `vsip_scalar_f`<br>`vsip_randu_f(`<br>`vsip_randstate * state);` | Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$. |
| `vsip_cscalar_f`<br>`vsip_crandu_f(`<br>`vsip_randstate * state);` | Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$. |
| `void`<br>`vsip_vrandu_f(`<br>`vsip_randstate * state,`<br>`const vsip_vview_f * R);` | 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 $(0, 2^{31} - 1)$. |
| `void`<br>`vsip_cvrandu_f(`<br>`vsip_randstate * state,`<br>`const vsip_cvview_f * R);` | 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 $(0, 2^{31} - 1)$. |
| `vsip_scalar_f`<br>`vsip_randn_f(`<br>`vsip_randstate * state);` | Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator. |
| `vsip_cscalar_f`<br>`vsip_crandn_f(`<br>`vsip_randstate * state);` | Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator. |

| Prototype | Description |
|---|---|
| `void vsip_vrandn_f( vsip_randstate * state, const vsip_vview_f * R);` | 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. |
| `void vsip_cvrandn_f( vsip_randstate * state, const vsip_cvview_f * R);` | 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. |

# Chapter 4. Elementwise Functions

## 4.1 Elementary Mathematical Functions

| Prototype | Description |
|---|---|
| `void`<br>`vsip_vacos_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a vector. |
| `void`<br>`vsip_vasin_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a vector. |
| `void`<br>`vsip_vatan_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a vector. |
| `void`<br>`vsip_vatan2_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * B,`<br>`const vsip_vview_f * R);` | Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input vectors. |
| `void`<br>`vsip_vcos_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Computes the cosine for each element of a vector. Element angle values are in radians. |
| `void`<br>`vsip_vexp_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Computes the exponential function value for each element of a vector. |
| `void`<br>`vsip_cvexp_f(`<br>`const vsip_cvview_f * A,`<br>`const vsip_cvview_f * R);` | Computes the exponential function value for each element of a vector. |
| `void`<br>`vsip_vexp10_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Computes the base 10 exponential for each element of a vector. |
| `void`<br>`vsip_vlog_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Computes the natural logarithm for each element of a vector. |
| `void`<br>`vsip_vlog10_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Compute the base ten logarithm for each element of a vector. |
| `void`<br>`vsip_vsin_f(`<br>`const vsip_vview_f * A,`<br>`const vsip_vview_f * R);` | Compute the sine for each element of a vector. Element angle values are in radians. |
| `void`<br>`vsip_Dvsqrt_P(`<br>`const vsip_Dvview_P * A,`<br>`const vsip_Dvview_P * R);` | Compute the square root for each element of a vector.<br>The following instances are supported:<br>`vsip_vsqrt_f`<br>`vsip_cvsqrt_f` |

| Prototype | Description |
|---|---|
| *void*<br>vsip_vtan_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * R); | Compute the tangent for each element of a vector. Element angle values are in radians. |

## 4.2 Unary Operations

| Prototype | Description |
|---|---|
| *void*<br>vsip_cvconj_f(<br>const vsip_cvview_f * A,<br>const vsip_cvview_f * R); | Compute the conjugate for each element of a complex vector. |
| *void*<br>vsip_veuler_f(<br>const vsip_vview_f * A,<br>const vsip_cvview_f * R); | Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a vector. |
| *void*<br>vsip_Dvmag_P(<br>const vsip_Dvview_P * A,<br>const vsip_vview_P * R); | Compute the magnitude for each element of a vector.<br>The following instances are supported:<br>vsip_vmag_f<br>vsip_vmag_i<br>vsip_cvmag_f |
| *void*<br>vsip_vcmagsq_f(<br>const vsip_cvview_f * A,<br>const vsip_vview_f * R); | Computes the square of the magnitudes for each element of a vector. |
| *vsip_Dscalar_P*<br>vsip_Dvmeanval_P(<br>const vsip_Dvview_P * A); | Returns the mean value of the elements of a vector.<br>The following instances are supported:<br>vsip_vmeanval_f<br>vsip_cvmeanval_f |
| *vsip_scalar_P*<br>vsip_Dvmeansqval_P(<br>const vsip_Dvview_P * A); | Returns the mean magnitude squared value of the elements of a vector.<br>The following instances are supported:<br>vsip_vmeansqval_f<br>vsip_cvmeansqval_f |
| *vsip_scalar_P*<br>vsip_Dvmodulate_P(<br>const vsip_Dvview_P * A,<br>const vsip_scalar_P nu,<br>const vsip_scalar_P phi,<br>const vsip_Dvview_P * R); | Computes the modulation of a real vector by a specified complex frequency.<br>The following instances are supported:<br>vsip_vmodulate_f<br>vsip_cvmodulate_f |
| *void*<br>vsip_Dvneg_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * R); | Computes the negation for each element of a vector.<br>The following instances are supported:<br>vsip_vneg_f<br>vsip_vneg_i<br>vsip_cvneg_f |

| Prototype | Description |
|---|---|
| *void*<br>vsip_Dvrecip_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * R); | Computes the reciprocal for each element of a vector.<br>The following instances are supported:<br>vsip_vrecip_f<br>vsip_cvrecip_f |
| *void*<br>vsip_vrsqrt_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * R); | Computes the reciprocal of the square root for each element of a vector. |
| *void*<br>vsip_vsq_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * R); | Computes the square for each element of a vector. |
| *vsip_scalar_P*<br>vsip_Dvsumval_P(<br>const vsip_vview_P * A); | Returns the sum of the elements of a vector.<br>The following instances are supported:<br>vsip_vsumval_f<br>vsip_vsumval_bl |
| *vsip_scalar_f*<br>vsip_vsumsqval_f(<br>const vsip_vview_f * A); | Returns the sum of the squares of the elements of a vector. |

## 4.3 Binary Operations

| Prototype | Description |
|---|---|
| *void*<br>vsip_Dvadd_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * R); | Computes the sum, by element, of two vectors.<br>The following instances are supported:<br>vsip_vadd_f<br>vsip_vadd_i<br>vsip_cvadd_f |
| *void*<br>vsip_rcvadd_f(<br>const vsip_vview_f * A,<br>const vsip_cvview_f * B,<br>const vsip_cvview_f * R); | Computes the sum, by element, of two vectors. |
| *void*<br>vsip_Dsvadd_P(<br>const vsip_Dscalar_P a,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * R); | Computes the sum, by element, of a scalar and a vector.<br>The following instances are supported:<br>vsip_svadd_f<br>vsip_svadd_i<br>vsip_csvadd_f |
| *void*<br>vsip_rscvadd_f(<br>const vsip_scalar_f a,<br>const vsip_cvview_f * B,<br>const vsip_cvview_f * R); | Computes the sum, by element, of a real scalar and a complex vector. |
| *void*<br>vsip_Dvdiv_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * R); | Computes the quotient, by element, of two vectors.<br>The following instances are supported:<br>vsip_vdiv_f<br>vsip_cvdiv_f |

| Prototype | Description |
|---|---|
| void<br>vsip_svdiv_f(<br>const vsip_scalar_f a,<br>const vsip_vview_f * B,<br>const vsip_vview_f * R); | Computes the quotient, by element, of a scalar and a vector. |
| void<br>vsip_rscvsub_f(<br>const vsip_scalar_f a,<br>const vsip_cvview_f * B,<br>const vsip_cvview_f * R); | Computes the difference, by element, of a real scalar and a complex vector. |
| void<br>vsip_Dvexpoavg_P(<br>const vsip_scalar_P a,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * C); | Computes an exponential weighted average, by element, of two vectors.<br>The following instances are supported:<br>vsip_vexpoavg_f<br>vsip_cvexpoavg_f |
| void<br>vsip_vhypot_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_f * R); | Computes the square root of the sum of squares, by element, of two input vectors. |
| void<br>vsip_cvjmul_f(<br>const vsip_cvview_f * A,<br>const vsip_cvview_f * B,<br>const vsip_cvview_f * R); | Computes the product of a complex vector with the conjugate of a second complex vector, by element. |
| void<br>vsip_Dvmul_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * R); | Computes the product, by element, of two vectors.<br>The following instances are supported:<br>vsip_vmul_f<br>vsip_vmul_i<br>vsip_cvmul_f |
| void<br>vsip_rcvmul_f(<br>const vsip_vview_f * A,<br>const vsip_cvview_f * B,<br>const vsip_cvview_f * R); | Computes the product, by element, of two vectors. |
| void<br>vsip_Dsvmul_P(<br>const vsip_Dscalar_P a,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * R); | Computes the product, by element, of a scalar and a vector.<br>The following instances are supported:<br>vsip_svmul_f<br>vsip_svmul_i<br>vsip_csvmul_f |
| void<br>vsip_rscvmul_f(<br>const vsip_scalar_f a,<br>const vsip_cvview_f * B,<br>const vsip_cvview_f * R); | Computes the product, by element, of a real scalar and a complex vector. |
| void<br>vsip_DvDmmul_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dmview_P * B,<br>const vsip_major major,<br>const vsip_Dmview_P * R); | Computes the product, by element, of a vector and the rows or columns of a matrix.<br>The following instances are supported:<br>vsip_vmmul_f<br>vsip_cvmmul_f |

| Prototype | Description |
|---|---|
| void<br>vsip_rvcmmul_f(<br>const vsip_vview_f * A,<br>const vsip_cmview_f * B,<br>const vsip_major major,<br>const vsip_cmview_f * R); | Computes the product, by element, of a vector and the rows or columns of a matrix. |
| void<br>vsip_Dvsub_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * R); | Computes the difference, by element, of two vectors.<br>The following instances are supported:<br>vsip_vsub_f<br>vsip_vsub_i<br>vsip_cvsub_f |
| void<br>vsip_crvsub_f(<br>const vsip_cvview_f * A,<br>const vsip_vview_f * B,<br>const vsip_cvview_f * R); | Computes the difference, by element, of two vectors. |
| void<br>vsip_rcvsub_f(<br>const vsip_vview_f * A,<br>const vsip_cvview_f * B,<br>const vsip_cvview_f * R); | Computes the difference, by element, of two vectors. |
| void<br>vsip_Dsvsub_P(<br>const vsip_Dscalar_P a,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * R); | Computes the difference, by element, of a scalar and a vector.<br>The following instances are supported:<br>vsip_svsub_f<br>vsip_svsub_i<br>vsip_csvsub_f |

## 4.4 Ternary Operations

| Prototype | Description |
|---|---|
| void<br>vsip_Dvam_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * C,<br>const vsip_Dvview_P * R); | Computes the sum of two vectors and product of a third vector, by element.<br>The following instances are supported:<br>vsip_vam_f<br>vsip_cvam_f |
| void<br>vsip_Dvma_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * C,<br>const vsip_Dvview_P * R); | Computes the product of two vectors and sum of a third vector, by element.<br>The following instances are supported:<br>vsip_vma_f<br>vsip_cvma_f |
| void<br>vsip_Dvmsa_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dscalar_P c,<br>const vsip_Dvview_P * R); | Computes the product of two vectors and sum of a scalar, by element.<br>The following instances are supported:<br>vsip_vmsa_f<br>vsip_cvmsa_f |

| Prototype | Description |
|---|---|
| *void*<br>vsip_Dvmsb_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * C,<br>const vsip_Dvview_P * R); | Computes the product of two vectors and difference of a third vector, by element.<br>The following instances are supported:<br>vsip_vmsb_f<br>vsip_cvmsb_f |
| *void*<br>vsip_Dvsam_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dscalar_P b,<br>const vsip_Dvview_P * C,<br>const vsip_Dvview_P * R); | Computes the sum of a vector and a scalar, and product with a second vector, by element.<br>The following instances are supported:<br>vsip_vsam_f<br>vsip_cvsam_f |
| *void*<br>vsip_Dvsbm_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B,<br>const vsip_Dvview_P * C,<br>const vsip_Dvview_P * R); | Computes the difference of two vectors, and product with a third vector, by element.<br>The following instances are supported:<br>vsip_vsbm_f<br>vsip_cvsbm_f |
| *void*<br>vsip_Dvsma_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dscalar_P b,<br>const vsip_Dvview_P * C,<br>const vsip_Dvview_P * R); | Computes the product of a vector and a scalar, and sum with a second vector, by element.<br>The following instances are supported:<br>vsip_vsma_f<br>vsip_cvsma_f |
| *void*<br>vsip_Dvsmsa_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dscalar_P b,<br>const vsip_Dscalar_P c,<br>const vsip_Dvview_P * R); | Computes the product of a vector and a scalar, and sum with a second scalar, by element.<br>The following instances are supported:<br>vsip_vsmsa_f<br>vsip_cvsmsa_f |

## 4.5 Logical Operations

| Prototype | Description |
|---|---|
| *vsip_scalar_bl*<br>vsip_valltrue_bl(<br>const vsip_vview_bl * A); | Returns true if all the elements of a vector are true. |
| *vsip_scalar_bl*<br>vsip_vanytrue_bl(<br>const vsip_vview_bl * A); | Returns true if one or more elements of a vector are true. |
| *void*<br>vsip_vleq_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_bl * R); | Computes the boolean comparison of 'equal', by element, of two vectors. |
| *void*<br>vsip_vlge_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_bl * R); | Computes the boolean comparison of 'greater than or equal', by element, of two vectors. |

| Prototype | Description |
|---|---|
| *void*<br>vsip_vlgt_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_bl * R); | Computes the boolean comparison of 'greater than', by element, of two vectors. |
| *void*<br>vsip_vlle_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_bl * R); | Computes the boolean comparison of 'less than or equal', by element, of two vectors. |
| *void*<br>vsip_vllt_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_bl * R); | Computes the boolean comparison of 'less than', by element, of two vectors. |
| *void*<br>vsip_vlne_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_bl * R); | Computes the boolean comparison of 'not equal', by element, of two vectors. |

## 4.6 Selection Operations

| Prototype | Description |
|---|---|
| *void*<br>vsip_vclip_P(<br>const vsip_vview_P * A,<br>const vsip_scalar_P t1,<br>const vsip_scalar_P t2,<br>const vsip_scalar_P c1,<br>const vsip_scalar_P c2,<br>const vsip_vview_P * R); | Computes the generalised double clip, by element, of two vectors.<br>The following instances are supported:<br>vsip_vclip_f<br>vsip_vclip_i |
| *void*<br>vsip_vinvclip_P(<br>const vsip_vview_P * A,<br>const vsip_scalar_P t1,<br>const vsip_scalar_P t2,<br>const vsip_scalar_P t3,<br>const vsip_scalar_P c1,<br>const vsip_scalar_P c2,<br>const vsip_vview_P * R); | Computes the generalised inverted double clip, by element, of two vectors.<br>The following instances are supported:<br>vsip_vinvclip_f<br>vsip_vinvclip_i |
| *vsip_length*<br>vsip_vindexbool(<br>const vsip_vview_bl * X,<br>vsip_vview_vi * Y); | Computes an index vector of the indices of the non-false elements of the boolean vector, and returns the number of non-false elements. |
| *void*<br>vsip_vmax_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_f * R); | Computes the maximum, by element, of two vectors. |
| *void*<br>vsip_vmaxmg_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_f * R); | Computes the maximum magnitude (absolute value), by element, of two vectors. |
| *void*<br>vsip_vcmaxmgsq_f(<br>const vsip_cvview_f * A,<br>const vsip_cvview_f * B,<br>const vsip_vview_f * R); | Computes the maximum magnitude squared, by element, of two complex vectors. |

| Prototype | Description |
|---|---|
| *vsip_scalar_f*<br>vsip_vcmaxmgsqval_f(<br>const vsip_cvview_f * A,<br>vsip_index * index); | 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. |
| *vsip_scalar_f*<br>vsip_vmaxmgval_f(<br>const vsip_vview_f * A,<br>vsip_index * index); | 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. |
| *vsip_scalar_f*<br>vsip_vmaxval_f(<br>const vsip_vview_f * A,<br>vsip_index * index); | 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. |
| *void*<br>vsip_vmin_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_f * R); | Computes the minimum, by element, of two vectors. |
| *void*<br>vsip_vminmg_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_vview_f * R); | Computes the minimum magnitude (absolute value), by element, of two vectors. |
| *void*<br>vsip_vcminmgsq_f(<br>const vsip_cvview_f * A,<br>const vsip_cvview_f * B,<br>const vsip_vview_f * R); | Computes the minimum magnitude squared, by element, of two complex vectors. |
| *vsip_scalar_f*<br>vsip_vcminmgsqval_f(<br>const vsip_cvview_f * A,<br>vsip_index * index); | 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. |
| *vsip_scalar_f*<br>vsip_vminmgval_f(<br>const vsip_vview_f * A,<br>vsip_index * index); | 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. |
| *vsip_scalar_f*<br>vsip_vminval_f(<br>const vsip_vview_f * A,<br>vsip_index * index); | 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 |
|---|---|
| *void*<br>vsip_vand_P(<br>const vsip_vview_P * A,<br>const vsip_vview_P * B,<br>const vsip_vview_P * R); | Computes the bitwise and, by element, of two vectors.<br>The following instances are supported:<br>vsip_vand_i<br>vsip_vand_bl |
| *void*<br>vsip_vnot_P(<br>const vsip_vview_P * A,<br>const vsip_vview_P * R); | Computes the bitwise not (one's complement), by element, of two vectors.<br>The following instances are supported:<br>vsip_vnot_i<br>vsip_vnot_bl |

| Prototype | Description |
|---|---|
| void<br>vsip_vor_P(<br>const vsip_vview_P * A,<br>const vsip_vview_P * B,<br>const vsip_vview_P * R); | Computes the bitwise inclusive or, by element, of two vectors.<br>The following instances are supported:<br>vsip_vor_i<br>vsip_vor_bl |
| void<br>vsip_vxor_P(<br>const vsip_vview_P * A,<br>const vsip_vview_P * B,<br>const vsip_vview_P * R); | Computes the bitwise exclusive or, by element, of two vectors.<br>The following instances are supported:<br>vsip_vxor_i<br>vsip_vxor_bl |

## 4.8 Element Generation and Copy

| Prototype | Description |
|---|---|
| void<br>vsip_Dvcopy_P_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * R); | Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.<br>The following instances are supported:<br>vsip_vcopy_f_f<br>vsip_vcopy_f_i<br>vsip_vcopy_f_bl<br>vsip_vcopy_i_f<br>vsip_vcopy_i_i<br>vsip_vcopy_i_vi<br>vsip_vcopy_bl_f<br>vsip_vcopy_bl_bl<br>vsip_vcopy_vi_i<br>vsip_vcopy_vi_vi<br>vsip_vcopy_mi_mi<br>vsip_cvcopy_f_f |
| void<br>vsip_Dmcopy_P_P(<br>const vsip_Dmview_P * A,<br>const vsip_Dmview_P * R); | Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.<br>The following instances are supported:<br>vsip_mcopy_f_f<br>vsip_cmcopy_f_f |
| void<br>vsip_Dvfill_P(<br>const vsip_Dscalar_P a,<br>const vsip_Dvview_P * R); | Fill a vector with a constant value.<br>The following instances are supported:<br>vsip_vfill_f<br>vsip_vfill_i<br>vsip_cvfill_f |
| void<br>vsip_vramp_P(<br>const vsip_scalar_P alpha,<br>const vsip_scalar_P beta,<br>const vsip_vview_P * R); | Computes a vector ramp by starting at an initial value and incrementing each successive element by the ramp step size.<br>The following instances are supported:<br>vsip_vramp_f<br>vsip_vramp_i |

## 4.9 Manipulation Operations

| Prototype | Description |
|---|---|
| *void*<br>vsip_vcmplx_f(<br>const vsip_vview_f * A,<br>const vsip_vview_f * B,<br>const vsip_cvview_f * R); | Form a complex vector from two real vectors. |
| *void*<br>vsip_Dvgather_P(<br>const vsip_Dvview_P * X,<br>const vsip_vview_vi * I,<br>const vsip_Dvview_P * Y); | 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.<br>The following instances are supported:<br>vsip_vgather_f<br>vsip_vgather_i<br>vsip_cvgather_f |
| *void*<br>vsip_vimag_f(<br>const vsip_cvview_f * A,<br>const vsip_vview_f * R); | Extract the imaginary part of a complex vector. |
| *void*<br>vsip_vpolar_f(<br>const vsip_cvview_f * A,<br>const vsip_vview_f * R,<br>const vsip_vview_f * P); | Convert a complex vector from rectangular to polar form. The polar data consists of a real vector containing the radius and a corresponding real vector containing the argument (angle) of the complex input data. |
| *void*<br>vsip_vreal_f(<br>const vsip_cvview_f * A,<br>const vsip_vview_f * R); | Extract the real part of a complex vector. |
| *void*<br>vsip_vrect_f(<br>const vsip_vview_f * R,<br>const vsip_vview_f * P,<br>const vsip_cvview_f * A); | Convert a pair of real vectors from complex polar to complex rectangular form. |
| *void*<br>vsip_Dvscatter_P(<br>const vsip_Dvview_P * X,<br>const vsip_Dvview_P * Y,<br>const vsip_vview_vi * I); | 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.<br>The following instances are supported:<br>vsip_vscatter_f<br>vsip_vscatter_i<br>vsip_cvscatter_f |
| *void*<br>vsip_Dvswap_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B); | Swap elements between two vectors.<br>The following instances are supported:<br>vsip_vswap_f<br>vsip_cvswap_f |

# Chapter 5. Signal Processing Functions

## 5.1 FFT Functions

| Prototype | Description |
|---|---|
| `vsip_fft_f *`<br>`vsip_ccfftip_create_f(`<br>`const vsip_index length,`<br>`const vsip_scalar_f scale,`<br>`const vsip_fft_dir dir,`<br>`const vsip_length ntimes,`<br>`const vsip_alg_hint hint);` | Create a 1D FFT object. |
| `vsip_fft_f *`<br>`vsip_ccfftop_create_f(`<br>`const vsip_index length,`<br>`const vsip_scalar_f scale,`<br>`const vsip_fft_dir dir,`<br>`const vsip_length ntimes,`<br>`const vsip_alg_hint hint);` | Create a 1D FFT object. |
| `vsip_fft_f *`<br>`vsip_crfftop_create_f(`<br>`const vsip_index length,`<br>`const vsip_scalar_f scale,`<br>`const vsip_length ntimes,`<br>`const vsip_alg_hint hint);` | Create a 1D FFT object. |
| `vsip_fft_f *`<br>`vsip_rcfftop_create_f(`<br>`const vsip_index length,`<br>`const vsip_scalar_f scale,`<br>`const vsip_length ntimes,`<br>`const vsip_alg_hint hint);` | Create a 1D FFT object. |
| `int`<br>`vsip_fft_destroy_f(`<br>`vsip_fft_f * plan);` | Destroy an FFT object. |
| `void`<br>`vsip_fft_getattr_f(`<br>`const vsip_fft_f * plan,`<br>`vsip_fft_attr_f * attr);` | Return the attributes of an FFT object. |
| `void`<br>`vsip_ccfftip_f(`<br>`const vsip_fft_f * plan,`<br>`const vsip_cvview_f * xy);` | Apply a complex-to-complex Fast Fourier Transform (FFT). |
| `void`<br>`vsip_ccfftop_f(`<br>`const vsip_fft_f * plan,`<br>`const vsip_cvview_f * x,`<br>`const vsip_cvview_f * y);` | Apply a complex-to-complex Fast Fourier Transform (FFT). |
| `void`<br>`vsip_crfftop_f(`<br>`const vsip_fft_f * plan,`<br>`const vsip_cvview_f * x,`<br>`const vsip_vview_f * y);` | Apply a complex-to-real Fast Fourier Transform (FFT). |
| `void`<br>`vsip_rcfftop_f(`<br>`const vsip_fft_f * plan,`<br>`const vsip_vview_f * x,`<br>`const vsip_cvview_f * y);` | Apply a real-to-complex Fast Fourier Transform (FFT). |

| Prototype | Description |
|---|---|
| vsip_fftm_f * vsip_ccfftmip_create_f( const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint); | Create a 1D multiple FFT object. |
| vsip_fftm_f * vsip_ccfftmop_create_f( const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint); | Create a 1D multiple FFT object. |
| vsip_fftm_f * vsip_crfftmop_create_f( const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint); | Create a 1D multiple FFT object. |
| vsip_fftm_f * vsip_rcfftmop_create_f( const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint); | Create a 1D multiple FFT object. |
| int vsip_fftm_destroy_f( vsip_fftm_f * plan); | Destroy an FFT object. |
| void vsip_fftm_getattr_f( const vsip_fftm_f * plan, vsip_fftm_attr_f * attr); | Return the attributes of an FFT object. |
| void vsip_ccfftmip_f( const vsip_fftm_f * plan, const vsip_cmview_f * XY); | Apply a multiple complex-to-complex Fast Fourier Transform (FFT). |
| void vsip_ccfftmop_f( const vsip_fftm_f * plan, const vsip_cmview_f * X, const vsip_cmview_f * Y); | Apply a multiple complex-to-complex Fast Fourier Transform (FFT). |
| void vsip_crfftmop_f( const vsip_fftm_f * plan, const vsip_cmview_f * X, const vsip_mview_f * Y); | Apply a multiple complex-to-real Fast Fourier Transform (FFT). |
| void vsip_rcfftmop_f( const vsip_fftm_f * plan, const vsip_mview_f * X, const vsip_cmview_f * Y); | Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT). |

## 5.2 Convolution/Correlation Functions

| Prototype | Description |
|---|---|
| *vsip_conv1d_f* *<br>vsip_conv1d_create_f(<br>const vsip_vview_f * kernel,<br>const vsip_symmetry symm,<br>const vsip_length N,<br>const vsip_length D,<br>const vsip_support_region support,<br>const vsip_length ntimes,<br>const vsip_alg_hint hint); | Create a decimated 1D convolution filter object. |
| *int*<br>vsip_conv1d_destroy_f(<br>vsip_conv1d_f * plan); | Destroy a 1D convolution object. |
| *void*<br>vsip_conv1d_getattr_f(<br>const vsip_conv1d_f * plan,<br>vsip_conv1d_attr_f * attr); | Returns the attributes for a 1D convolution object. |
| *void*<br>vsip_convolve1d_f(<br>const vsip_conv1d_f * plan,<br>const vsip_vview_f * x,<br>const vsip_vview_f * y); | Compute a decimated real one-dimensional (1D) convolution of two vectors. |
| *vsip_Dcorr1d_P* *<br>vsip_Dcorr1d_create_P(<br>const vsip_length M,<br>const vsip_length N,<br>const vsip_support_region support,<br>const vsip_length ntimes,<br>const vsip_alg_hint hint); | Create a 1D correlation object.<br>The following instances are supported:<br>vsip_corr1d_create_f<br>vsip_ccorr1d_create_f |
| *int*<br>vsip_Dcorr1d_destroy_P(<br>vsip_Dcorr1d_P * plan); | Destroy a 1D correlation object.<br>The following instances are supported:<br>vsip_corr1d_destroy_f<br>vsip_ccorr1d_destroy_f |
| *void*<br>vsip_Dcorr1d_getattr_P(<br>const vsip_Dcorr1d_P * plan,<br>vsip_Dcorr1d_attr_P * attr); | Return the attributes for a 1D correlation object.<br>The following instances are supported:<br>vsip_corr1d_getattr_f<br>vsip_ccorr1d_getattr_f |
| *void*<br>vsip_Dcorrelate1d_P(<br>const vsip_Dcorr1d_P * plan,<br>const vsip_bias bias,<br>const vsip_Dvview_P * ref,<br>const vsip_Dvview_P * x,<br>const vsip_Dvview_P * y); | Compute a real one-dimensional (1D) correlation of two vectors.<br>The following instances are supported:<br>vsip_correlate1d_f<br>vsip_ccorrelate1d_f |

## 5.3 Window Functions

| Prototype | Description |
|---|---|
| *vsip_vview_f* *<br>vsip_vcreate_blackman_f(<br>const vsip_length N,<br>const vsip_memory_hint hint); | Create a vector with Blackman window weights. |

| Prototype | Description |
|---|---|
| *vsip_vview_f* * vsip_vcreate_cheby_f( const vsip_length N, const vsip_scalar_f ripple, const vsip_memory_hint hint); | Create a vector with Dolph-Chebyshev window weights. |
| *vsip_vview_f* * vsip_vcreate_hanning_f( const vsip_length N, const vsip_memory_hint hint); | Create a vector with Hanning window weights. |
| *vsip_vview_f* * vsip_vcreate_kaiser_f( const vsip_length N, const vsip_scalar_f beta, const vsip_memory_hint hint); | Create a vector with Kaiser window weights. |

## 5.4 Filter Functions

| Prototype | Description |
|---|---|
| *vsip_Dfir_P* * vsip_Dfir_create_P( const vsip_Dvview_P * kernel, const vsip_symmetry symm, const vsip_length N, const vsip_length D, const vsip_obj_state state, const vsip_length ntimes, const vsip_alg_hint hint); | Create a decimated FIR filter object. The following instances are supported: vsip_fir_create_f vsip_cfir_create_f |
| *int* vsip_Dfir_destroy_P( vsip_Dfir_P * plan); | Destroy a FIR filter object. The following instances are supported: vsip_fir_destroy_f vsip_cfir_destroy_f |
| *int* vsip_Dfirflt_P( vsip_Dfir_P * plan, const vsip_Dvview_P * x, const vsip_Dvview_P * y); | FIR filter an input sequence and decimate the output. The following instances are supported: vsip_firflt_f vsip_cfirflt_f |
| *void* vsip_Dfir_getattr_P( const vsip_Dfir_P * plan, vsip_Dfir_attr_P * attr); | Return the attributes of a FIR filter object. The following instances are supported: vsip_fir_getattr_f vsip_cfir_getattr_f |

## 5.5 Miscellaneous Signal Processing Functions

| Prototype | Description |
|---|---|
| *void* vsip_vhisto_f( const vsip_vview_f * A, const vsip_scalar_f min, const vsip_scalar_f max, const vsip_hist_opt opt, const vsip_vview_f * R); | Compute the histogram of a vector. |

# Chapter 6. Linear Algebra

## 6.1 Matrix and Vector Operations

| Prototype | Description |
|---|---|
| `void`<br>`vsip_cmherm_f(`<br>`const vsip_cmview_f * A,`<br>`const vsip_cmview_f * R);` | Complex Hermitian (conjugate transpose) of a matrix. |
| `vsip_cscalar_f`<br>`vsip_cvjdot_f(`<br>`const vsip_cvview_f * A,`<br>`const vsip_cvview_f * B);` | Compute the conjugate inner (dot) product of two complex vectors. |
| `void`<br>`vsip_gemp_f(`<br>`const vsip_scalar_f alpha,`<br>`const vsip_mview_f * A,`<br>`const vsip_mat_op Aop,`<br>`const vsip_mview_f * B,`<br>`const vsip_mat_op Bop,`<br>`const vsip_scalar_f beta,`<br>`const vsip_mview_f * R);` | Calculate the general product of two matrices and accumulate. |
| `void`<br>`vsip_cgemp_f(`<br>`const vsip_cscalar_f alpha,`<br>`const vsip_cmview_f * A,`<br>`const vsip_mat_op Aop,`<br>`const vsip_cmview_f * B,`<br>`const vsip_mat_op Bop,`<br>`const vsip_cscalar_f beta,`<br>`const vsip_cmview_f * R);` | Calculate the general product of two matrices and accumulate. |
| `void`<br>`vsip_gems_f(`<br>`const vsip_scalar_f alpha,`<br>`const vsip_mview_f * A,`<br>`const vsip_mat_op Aop,`<br>`const vsip_scalar_f beta,`<br>`const vsip_mview_f * C);` | Calculate a general matrix sum. |
| `void`<br>`vsip_cgems_f(`<br>`const vsip_cscalar_f alpha,`<br>`const vsip_cmview_f * A,`<br>`const vsip_mat_op Aop,`<br>`const vsip_cscalar_f beta,`<br>`const vsip_cmview_f * C);` | Calculate a general matrix sum. |
| `void`<br>`vsip_Dmprod_P(`<br>`const vsip_Dmview_P * A,`<br>`const vsip_Dmview_P * B,`<br>`const vsip_Dmview_P * R);` | Calculate the product of two matrices. The following instances are supported:<br>`vsip_mprod_f`<br>`vsip_cmprod_f` |
| `void`<br>`vsip_cmprodh_f(`<br>`const vsip_cmview_f * A,`<br>`const vsip_cmview_f * B,`<br>`const vsip_cmview_f * R);` | Calculate the product a complex matrix and the Hermitian of a complex matrix. |

| Prototype | Description |
|---|---|
| void<br>vsip_cmprodj_f(<br>const vsip_cmview_f * A,<br>const vsip_cmview_f * B,<br>const vsip_cmview_f * R); | Calculate the product a complex matrix and the conjugate of a complex matrix. |
| void<br>vsip_Dmprodt_P(<br>const vsip_Dmview_P * A,<br>const vsip_Dmview_P * B,<br>const vsip_Dmview_P * R); | Calculate the product of a matrix and the transpose of a matrix.<br>The following instances are supported:<br>vsip_mprodt_f<br>vsip_cmprodt_f |
| void<br>vsip_Dmvprod_P(<br>const vsip_Dmview_P * A,<br>const vsip_Dvview_P * X,<br>const vsip_Dvview_P * Y); | Calculate a matrix–vector product.<br>The following instances are supported:<br>vsip_mvprod_f<br>vsip_cmvprod_f |
| void<br>vsip_Dmtrans_P(<br>const vsip_Dmview_P * A,<br>const vsip_Dmview_P * R); | Transpose a matrix.<br>The following instances are supported:<br>vsip_mtrans_f<br>vsip_cmtrans_f |
| vsip_Dscalar_P<br>vsip_Dvdot_P(<br>const vsip_Dvview_P * A,<br>const vsip_Dvview_P * B); | Compute the inner (dot) product of two vectors.<br>The following instances are supported:<br>vsip_vdot_f<br>vsip_cvdot_f |
| void<br>vsip_Dvmprod_P(<br>const vsip_Dvview_P * X,<br>const vsip_Dmview_P * A,<br>const vsip_Dvview_P * Y); | Calculate a vector–matrix product.<br>The following instances are supported:<br>vsip_vmprod_f<br>vsip_cvmprod_f |
| void<br>vsip_vouter_f(<br>const vsip_scalar_f alpha,<br>const vsip_vview_f * X,<br>const vsip_vview_f * Y,<br>const vsip_vview_f * R); | Calculate the outer product of two vectors. |
| void<br>vsip_cvouter_f(<br>const vsip_cscalar_f alpha,<br>const vsip_cvview_f * X,<br>const vsip_cvview_f * Y,<br>const vsip_cvview_f * R); | Calculate the outer product of two vectors. |

## 6.2 Special Linear System Solvers

| Prototype | Description |
|---|---|
| int<br>vsip_covsol_f(<br>const vsip_mview_f * A,<br>const vsip_mview_f * XB); | Solve a covariance linear system problem. |
| int<br>vsip_ccovsol_f(<br>const vsip_cmview_f * A,<br>const vsip_cmview_f * XB); | Solve a covariance linear system problem. |

| Prototype | Description |
|---|---|
| `int`<br>`vsip_llsqsol_f(`<br>`const vsip_mview_f * A,`<br>`const vsip_mview_f * XB);` | Solve a linear least squares problem. |
| `int`<br>`vsip_cllsqsol_f(`<br>`const vsip_cmview_f * A,`<br>`const vsip_cmview_f * XB);` | Solve a linear least squares problem. |
| `int`<br>`vsip_toepsol_f(`<br>`const vsip_vview_f * T,`<br>`const vsip_vview_f * B,`<br>`const vsip_vview_f * W,`<br>`const vsip_vview_f * X);` | Solve a real symmetric positive definite Toeplitz linear system. |
| `int`<br>`vsip_ctoepsol_f(`<br>`const vsip_cvview_f * T,`<br>`const vsip_cvview_f * B,`<br>`const vsip_cvview_f * W,`<br>`const vsip_cvview_f * X);` | Solve a complex Hermitian positive definite Toeplitz linear system. |

## 6.3 General Square Linear System Solver

| Prototype | Description |
|---|---|
| *int*<br>vsip_Dlud_P(<br>vsip_clu_P * lud,<br>const vsip_Dmview_P * A); | Compute an LU decomposition of a square matrix using partial pivoting.<br>The following instances are supported:<br>vsip_lud_f<br>vsip_clud_f |
| *vsip_Dlu_P **<br>vsip_Dlud_create_P(<br>const vsip_length N); | Create an LU decomposition object.<br>The following instances are supported:<br>vsip_lud_create_f<br>vsip_clud_create_f |
| *int*<br>vsip_Dlud_destroy_P(<br>vsip_Dlu_P * lud); | Destroy an LU decomposition object.<br>The following instances are supported:<br>vsip_lud_destroy_f<br>vsip_clud_destroy_f |
| *void*<br>vsip_Dlud_getattr_P(<br>const vsip_Dlu_P * lud,<br>vsip_Dlu_attr_P * attr); | Returns the attributes of an LU decomposition object.<br>The following instances are supported:<br>vsip_lud_getattr_f<br>vsip_clud_getattr_f |
| *int*<br>vsip_lusol_f(<br>const vsip_clu_f * clud,<br>const vsip_mat_op opA,<br>const vsip_mview_f * XB); | Solve a square linear system. |
| *int*<br>vsip_clusol_f(<br>const vsip_clu_f * clud,<br>const vsip_mat_op opA,<br>const vsip_cmview_f * XB); | Solve a square linear system. |

## 6.4 Symmetric Positive Definite Linear System Solver

| Prototype | Description |
|---|---|
| *int*<br>vsip_chold_f(<br>vsip_cchol_f * chold,<br>const vsip_mview_f * A); | Compute a Cholesky decomposition of a symmetric positive definite matrix. |
| *int*<br>vsip_cchold_f(<br>vsip_cchol_f * chold,<br>const vsip_cmview_f * A); | Compute a Cholesky decomposition of a Hermitian positive definite matrix. |
| *vsip_chol_f **<br>vsip_chold_create_f(<br>const vsip_mat_uplo uplo,<br>const vsip_length n); | Creates a Cholesky decomposition object. |
| *vsip_cchol_f **<br>vsip_cchold_create_f(<br>const vsip_mat_uplo uplo,<br>const vsip_length n); | Creates a Cholesky decomposition object. |

| Prototype | Description |
|---|---|
| *int*<br>vsip_Dchold_destroy_P(<br>vsip_Dchol_P * chold); | Destroy a Cholesky decomposition object.<br>The following instances are supported:<br>vsip_chold_destroy_f<br>vsip_cchold_destroy_f |
| *void*<br>vsip_Dchold_getattr_P(<br>const vsip_Dchol_P * chold,<br>vsip_Dchol_attr_P * attr); | Returns the attributes of a Cholesky decomposition object.<br>The following instances are supported:<br>vsip_chold_getattr_f<br>vsip_cchold_getattr_f |
| *int*<br>vsip_cholsol_f(<br>const vsip_cchol_f * chold,<br>const vsip_mview_f * XB); | Solve a symmetric positive definite linear system. |
| *int*<br>vsip_ccholsol_f(<br>const vsip_cchol_f * chold,<br>const vsip_cmview_f * XB); | Solve a Hermitian positive definite linear system. |

## 6.5 Overdetermined Linear System Solver

| Prototype | Description |
|---|---|
| *int*<br>vsip_qrd_f(<br>vsip_cqr_f * qrd,<br>const vsip_mview_f * A); | Compute a QR decomposition of a matrix . |
| *int*<br>vsip_cqrd_f(<br>vsip_cqr_f * qrd,<br>const vsip_cmview_f * A); | Compute a QR decomposition of a matrix . |
| *vsip_qr_f **<br>vsip_qrd_create_f(<br>const vsip_length m,<br>const vsip_length n,<br>const vsip_qrd_qopt qopt); | Create a QR decomposition object. |
| *vsip_cqr_f **<br>vsip_cqrd_create_f(<br>const vsip_length m,<br>const vsip_length n,<br>const vsip_qrd_qopt qopt); | Create a QR decomposition object. |
| *int*<br>vsip_Dqrd_destroy_P(<br>vsip_Dqr_P * qrd); | Destroy a QR decomposition object.<br>The following instances are supported:<br>vsip_qrd_destroy_f<br>vsip_cqrd_destroy_f |
| *void*<br>vsip_Dqrd_getattr_P(<br>const vsip_Dqr_P * qrd,<br>vsip_Dqr_attr_P * attr); | Returns the attributes of a QR decomposition object.<br>The following instances are supported:<br>vsip_qrd_getattr_f<br>vsip_cqrd_getattr_f |

| Prototype | Description |
|---|---|
| `int vsip_qrdprodq_f( const vsip_qr_f * qrd, const vsip_mat_op opQ, const vsip_mat_side apQ, const vsip_mview_f * C);` | Multiply a matrix by the matrix $Q$ from a QR decomposition. |
| `int vsip_cqrdprodq_f( const vsip_cqr_f * qrd, const vsip_mat_op opQ, const vsip_mat_side apQ, const vsip_cmview_f * C);` | Multiply a matrix by the matrix $Q$ from a QR decomposition. |
| `int vsip_qrdsolr_f( const vsip_qr_f * qrd, const vsip_mat_op OpR, const vsip_scalar_f alpha, const vsip_mview_f * XB);` | Solve linear system based on the matrix $R$, from QR decomposition of the matrix $A$. |
| `int vsip_cqrdsolr_f( const vsip_cqr_f * qrd, const vsip_mat_op OpR, const vsip_cscalar_f alpha, const vsip_cmview_f * XB);` | Solve linear system based on the matrix $R$, from QR decomposition of the matrix $A$. |
| `int vsip_qrsol_f( const vsip_qr_f * qrd, const vsip_qrd_prob prob, const vsip_mview_f * XB);` | Solve either a linear covariance or linear least squares problem. |
| `int vsip_cqrsol_f( const vsip_cqr_f * qrd, const vsip_qrd_prob prob, const vsip_cmview_f * XB);` | Solve either a linear covariance or linear least squares problem. |