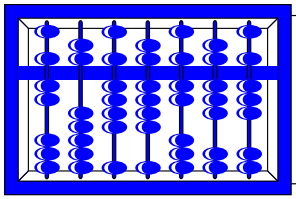

NASoftware Limited
Wynberg House, 1 Prospect Road, Prenton, CH42 8LE. UK
Tel: +44 151 609 1911 Fax: +44 151 550 7830

VSIPL
List of Contents

VSIPL/Contents [2.0]

Release 2.0
June 2009

Chapter 1. Support Functions

1.1 Initilialisation and Finalisation

Prototype	Description
int vsip_init(void * ptr);	Provides initialisation, allowing the library to allocate and set a global state, and prepare to support the use of VSIPL functionality by the user.
int vsip_finalize(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 Thread_SetParams(vsip_length num_threads, vsip_length max_num_running);	Allows the library to allocate a number of threads.

1.2 Array and Block Functions

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

Prototype	Description
vsip_Dblock_P * vsip_Dblockcreate_P(vsip_length num_items, vsip_memory_hint hint);	Creates a VSIPL block and binds a (VSIPL-allocated) data array to it. The following instances are supported: vsip_blockcreate_f vsip_blockcreate_i vsip_blockcreate_si vsip_cblockcreate_f vsip_cblockcreate_i vsip_cblockcreate_si vsip_blockcreate_bl vsip_blockcreate_vi vsip_blockcreate_mi
void vsip_Dblockdestroy_P(vsip_Dblock_P * block);	Destroy a VSIPL block object and any memory allocated for it by VSIPL. The following instances are supported: vsip_blockdestroy_f vsip_blockdestroy_i vsip_blockdestroy_si vsip_cblockdestroy_f vsip_cblockdestroy_i vsip_cblockdestroy_si vsip_blockdestroy_bl vsip_blockdestroy_vi vsip_blockdestroy_mi
vsip_scalar_P * vsip_blockfind_P(const vsip_block_P * block);	Find the pointer to the data bound to a VSIPL released block object. The following instances are supported: vsip_blockfind_f vsip_blockfind_i vsip_blockfind_si vsip_blockfind_bl vsip_blockfind_vi vsip_blockfind_mi
void vsip_cblockfind_P(const vsip_cblock_P * block, vsip_scalar_P ** user_data1, vsip_scalar_P ** user_data2);	Find the pointer(s) to the data bound to a VSIPL released complex block object. The following instances are supported: vsip_cblockfind_f vsip_cblockfind_i vsip_cblockfind_si
vsip_scalar_P * vsip_blockrebind_P(vsip_block_P * block, vsip_scalar_P * new_data);	Rebind a VSIPL block to user-specified data. The following instances are supported: vsip_blockrebind_f vsip_blockrebind_i vsip_blockrebind_si vsip_blockrebind_bl vsip_blockrebind_vi vsip_blockrebind_mi

Prototype	Description
void vsip_cblockrebind_P(vsip_cblock_P * block, vsip_scalar_P * new_data1, vsip_scalar_P * new_data2, vsip_scalar_P ** old_data1, vsip_scalar_P ** old_data2);	Rebind a VSIPL complex block to user-specified data. The following instances are supported: vsip_cblockrebind_f vsip_cblockrebind_i vsip_cblockrebind_si
*vsip_scalar_P ** vsip_blockrelease_P(vsip_block_P * block, vsip_scalar_bl update);	Release a VSIPL block for direct user access. The following instances are supported: vsip_blockrelease_f vsip_blockrelease_i vsip_blockrelease_si vsip_blockrelease_bl vsip_blockrelease_vi vsip_blockrelease_mi
void vsip_cblockrelease_P(vsip_cblock_P * block, vsip_scalar_bl update, vsip_scalar_P ** user_data1, vsip_scalar_P ** user_data2);	Release a complex block from VSIPL for direct user access. The following instances are supported: vsip_cblockrelease_f vsip_cblockrelease_i vsip_cblockrelease_si
void vsip_complete(void);	Force all deferred VSIPL execution to complete.
vsip_cmplx_mem vsip_cstorage(void);	Returns the preferred complex storage format for the system.

1.3 Vector View Functions

Prototype	Description
void vsip_Dvalldestroy_P(vsip_Dvview_P * vector);	Destroy a vector, its associated block, and any VSIPL data array bound to the block. The following instances are supported: vsip_valldestroy_f vsip_valldestroy_i vsip_valldestroy_si vsip_cvalldestroy_f vsip_cvalldestroy_i vsip_cvalldestroy_si vsip_valldestroy_bl vsip_valldestroy_vi vsip_valldestroy_mi

Prototype	Description
vsip_Dvview_P * vsip_Dvbind_P(vsip_Dblock_P * block, vsip_offset offset, vsip_stride stride, vsip_length length);	Create a vector view object and bind it to a block object. The following instances are supported: vsip_vbind_f vsip_vbind_i vsip_vbind_si vsip_cvbind_f vsip_cvbind_i vsip_cvbind_si vsip_vbind_bl vsip_vbind_vi vsip_vbind_mi
vsip_Dvview_P * vsip_Dvcloneview_P(const vsip_Dvview_P * vector);	Create a clone of a vector view. The following instances are supported: vsip_vcloneview_f vsip_vcloneview_i vsip_vcloneview_si vsip_cvcloneview_f vsip_cvcloneview_i vsip_cvcloneview_si vsip_vcloneview_bl vsip_vcloneview_vi vsip_vcloneview_mi
vsip_Dvview_P * vsip_Dvcreate_P(vsip_length length, vsip_memory_hint hint);	Creates a block object and a vector view object of the block. The following instances are supported: vsip_vcreate_f vsip_vcreate_i vsip_vcreate_si vsip_cvcreate_f vsip_cvcreate_i vsip_cvcreate_si vsip_vcreate_bl vsip_vcreate_vi vsip_vcreate_mi
vsip_Dblock_P * vsip_Dvdestroy_P(vsip_Dvview_P * vector);	Destroy a vector view object and return a pointer to the associated block object. The following instances are supported: vsip_vdestroy_f vsip_vdestroy_i vsip_vdestroy_si vsip_cvdestroy_f vsip_cvdestroy_i vsip_cvdestroy_si vsip_vdestroy_bl vsip_vdestroy_vi vsip_vdestroy_mi

Prototype	Description
vsip_Dscalar_P vsip_Dvget_P(const vsip_Dvview_P * vector, vsip_index j);	Get the value of a specified element of a vector view object. The following instances are supported: vsip_vget_f vsip_vget_i vsip_vget_si vsip_cvget_f vsip_cvget_i vsip_cvget_si vsip_vget_bl vsip_vget_vi vsip_vget_mi
void vsip_Dvgetattrib_P(const vsip_Dvview_P * vector, vsip_Dvattr_P * attr);	Return the attributes of a vector view object. The following instances are supported: vsip_vgetattrib_f vsip_vgetattrib_i vsip_vgetattrib_si vsip_cvgetattrib_f vsip_cvgetattrib_i vsip_cvgetattrib_si vsip_vgetattrib_bl vsip_vgetattrib_vi vsip_vgetattrib_mi
vsip_Dblock_P * vsip_Dvgetblock_P(const vsip_Dvview_P * vector);	Get the block attribute of a vector view object. The following instances are supported: vsip_vgetblock_f vsip_vgetblock_i vsip_vgetblock_si vsip_cvgetblock_f vsip_cvgetblock_i vsip_cvgetblock_si vsip_vgetblock_bl vsip_vgetblock_vi vsip_vgetblock_mi
vsip_length vsip_Dvgetlength_P(const vsip_Dvview_P * vector);	Get the length attribute of a vector view object. The following instances are supported: vsip_vgetlength_f vsip_vgetlength_i vsip_vgetlength_si vsip_cvgetlength_f vsip_cvgetlength_i vsip_cvgetlength_si vsip_vgetlength_bl vsip_vgetlength_vi vsip_vgetlength_mi

Prototype	Description
vsip_offset vsip_Dvgetoffset_P(const vsip_Dvview_P * vector);	Get the offset attribute of a vector view object. The following instances are supported: vsip_vgetoffset_f vsip_vgetoffset_i vsip_vgetoffset_si vsip_cvgetoffset_f vsip_cvgetoffset_i vsip_cvgetoffset_si vsip_vgetoffset_bl vsip_vgetoffset_vi vsip_vgetoffset_mi
vsip_stride vsip_Dvgetstride_P(const vsip_Dvview_P * vector);	Get the stride attribute of a vector view object. The following instances are supported: vsip_vgetstride_f vsip_vgetstride_i vsip_vgetstride_si vsip_cvgetstride_f vsip_cvgetstride_i vsip_cvgetstride_si vsip_vgetstride_bl vsip_vgetstride_vi vsip_vgetstride_mi
vsip_vview_P * vsip_vimagview_P(const vsip_cvview_P * complex_vector);	Create a vector view object of the imaginary part of a complex vector from a complex vector view object. The following instances are supported: vsip_vimagview_f vsip_vimagview_i vsip_vimagview_si
void vsip_Dvput_P(vsip_Dvview_P * vector, vsip_index j, vsip_Dscalar_P value);	Set the value of a specified element of a vector view object. The following instances are supported: vsip_vput_f vsip_vput_i vsip_vput_si vsip_cvput_f vsip_cvput_i vsip_cvput_si vsip_vput_bl vsip_vput_vi vsip_vput_mi
vsip_Dvview_P * vsip_Dvputattrib_P(vsip_Dvview_P * ve3tor, const vsip_Dvattr_P * attr);	Set the attributes of a vector view object. The following instances are supported: vsip_vputattrib_f vsip_vputattrib_i vsip_vputattrib_si vsip_cvputattrib_f vsip_cvputattrib_i vsip_cvputattrib_si vsip_vputattrib_bl vsip_vputattrib_vi vsip_vputattrib_mi

Prototype	Description
vsip_Dvview_P * vsip_Dvputlength_P(vsip_Dvview_P * vector, vsip_length length);	Set the length attribute of a vector view object. The following instances are supported: vsip_vputlength_f vsip_vputlength_i vsip_vputlength_si vsip_cvputlength_f vsip_cvputlength_i vsip_cvputlength_si vsip_vputlength_bl vsip_vputlength_vi vsip_vputlength_mi
vsip_Dvview_P * vsip_Dvputoffset_P(vsip_Dvview_P * vector, vsip_offset offset);	Set the offset attribute of a vector view object. The following instances are supported: vsip_vputoffset_f vsip_vputoffset_i vsip_vputoffset_si vsip_cvputoffset_f vsip_cvputoffset_i vsip_cvputoffset_si vsip_vputoffset_bl vsip_vputoffset_vi 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. The following instances are supported: vsip_vputstride_f vsip_vputstride_i vsip_vputstride_si vsip_cvputstride_f vsip_cvputstride_i vsip_cvputstride_si vsip_vputstride_bl vsip_vputstride_vi vsip_vputstride_mi
vsip_vview_P * vsip_vrealview_P(const vsip_cvview_P * complex_vector);	Create a vector view object of the real part of a complex vector from a complex vector view object. The following instances are supported: vsip_vrealview_f vsip_vrealview_i vsip_vrealview_si
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. The following instances are supported: vsip_vsubview_f vsip_vsubview_i vsip_vsubview_si vsip_cvsubview_f vsip_cvsubview_i vsip_cvsubview_si vsip_vsubview_bl vsip_vsubview_vi 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. The following instances are supported: vsip_malldestroy_f vsip_malldestroy_i vsip_malldestroy_si vsip_cmalldestroy_f vsip_cmalldestroy_i vsip_cmalldestroy_si vsip_malldestroy_bl
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. The following instances are supported: vsip_mbind_f vsip_mbind_i vsip_mbind_si vsip_cmbind_f vsip_cmbind_i vsip_cmbind_si vsip_mbind_bl
vsip_Dmview_P * vsip_Dmcloneview_P(const vsip_Dmview_P * matrix);	Create a clone of a matrix view. The following instances are supported: vsip_mcloneview_f vsip_mcloneview_i vsip_mcloneview_si vsip_cmcloneview_f vsip_cmcloneview_i vsip_cmcloneview_si vsip_mcloneview_bl
vsip_Dvview_P * vsip_Dmcolview_P(const vsip_Dmview_P * matrix, vsip_index j);	Create a vector view object of a specified column of the source matrix view object. The following instances are supported: vsip_mcolview_f vsip_mcolview_i vsip_mcolview_si vsip_cmcolview_f vsip_cmcolview_i vsip_cmcolview_si vsip_mcolview_bl
vsip_Dmview_P * vsip_Dmcreate_P(vsip_length col_length, vsip_length row_length, vsip_major rc, vsip_memory_hint hint);	Creates a block object and matrix view object of the block. The following instances are supported: vsip_mcreate_f vsip_mcreate_i vsip_mcreate_si vsip_cmcreate_f vsip_cmcreate_i vsip_cmcreate_si vsip_mcreate_bl

Prototype	Description
vsip_Dblock_P * vsip_Dmdestroy_P(vsip_Dmview_P * matrix);	Destroy a matrix view object and returns a pointer to the associated block object. The following instances are supported: vsip_mdestroy_f vsip_mdestroy_i vsip_mdestroy_si vsip_cmdestroy_f vsip_cmdestroy_i vsip_cmdestroy_si vsip_mdestroy_bl
vsip_Dvview_P * vsip_Dmdiagview_P(const vsip_Dmview_P * matrix, vsip_stride index);	Create a vector view object of a matrix diagonal of a matrix view object. The following instances are supported: vsip_mdiagview_f vsip_mdiagview_i vsip_mdiagview_si vsip_cmdiagview_f vsip_cmdiagview_i vsip_cmdiagview_si vsip_mdiagview_bl
vsip_Dscalar_P vsip_Dmget_P(const vsip_Dmview_P * matrix, vsip_index i, vsip_index j);	Get the value of a specified element of a matrix view object. The following instances are supported: vsip_mget_f vsip_mget_i vsip_mget_si vsip_cmget_f vsip_cmget_i vsip_cmget_si vsip_mget_bl
void vsip_Dmgetattrib_P(const vsip_Dmview_P * matrix, vsip_Dmattr_P * attr);	Get the attributes of a matrix view object. The following instances are supported: vsip_mgetattrib_f vsip_mgetattrib_i vsip_mgetattrib_si vsip_cmgetattrib_f vsip_cmgetattrib_i vsip_cmgetattrib_si vsip_mgetattrib_bl
vsip_Dblock_P * vsip_Dmgetblock_P(const vsip_Dmview_P * matrix);	Get the block attribute of a matrix view object. The following instances are supported: vsip_mgetblock_f vsip_mgetblock_i vsip_mgetblock_si vsip_cmgetblock_f vsip_cmgetblock_i vsip_cmgetblock_si vsip_mgetblock_bl

Prototype	Description
vsip_length vsip_Dmgetcollength_P(const vsip_Dmview_P * matrix);	Get the column length attribute of a matrix view object. The following instances are supported: vsip_mgetcollength_f vsip_mgetcollength_i vsip_mgetcollength_si vsip_cmgetcollength_f vsip_cmgetcollength_i vsip_cmgetcollength_si vsip_mgetcollength_bl
vsip_stride vsip_Dmgetcolstride_P(const vsip_Dmview_P * matrix);	Get the column stride attribute of a matrix view object. The following instances are supported: vsip_mgetcolstride_f vsip_mgetcolstride_i vsip_mgetcolstride_si vsip_cmgetcolstride_f vsip_cmgetcolstride_i vsip_cmgetcolstride_si vsip_mgetcolstride_bl
vsip_offset vsip_Dmgetoffset_P(const vsip_Dmview_P * matrix);	Get the offset attribute of a matrix view object. The following instances are supported: vsip_mgetoffset_f vsip_mgetoffset_i vsip_mgetoffset_si vsip_cmgetoffset_f vsip_cmgetoffset_i vsip_cmgetoffset_si vsip_mgetoffset_bl
vsip_length vsip_Dmgetrowlength_P(const vsip_Dmview_P * matrix);	Get the row length attribute of a matrix view object. The following instances are supported: vsip_mgetrowlength_f vsip_cmgetrowlength_f vsip_cmgetrowlength_i vsip_cmgetrowlength_si
vsip_stride vsip_Dmgetrowstride_P(const vsip_Dmview_P * matrix);	Get the row stride attribute of a matrix view object. The following instances are supported: vsip_mgetrowstride_f vsip_mgetrowstride_i vsip_mgetrowstride_si vsip_cmgetrowstride_f vsip_cmgetrowstride_i vsip_cmgetrowstride_si vsip_mgetrowstride_bl
vsip_mview_P * vsip_mimagview_P(const vsip_cmview_P * complex_matrix);	Create a matrix view object of the imaginary part of complex matrix from a complex matrix view object. The following instances are supported: vsip_mimagview_f vsip_mimagview_i vsip_mimagview_si

Prototype	Description
void vsip_Dmput_P(const vsip_Dmview_P * matrix, vsip_index i, vsip_index j, vsip_Dscalar_P value);	Set the value of a specified element of a matrix view object. The following instances are supported: vsip_mput_f vsip_mput_i vsip_mput_si vsip_cmput_f vsip_cmput_i vsip_cmput_si vsip_mput_bl
vsip_Dmview_P * vsip_Dmputattrib_P(vsip_Dmview_P * matrix, const vsip_Dmattr_P * attr);	Set the attributes of a matrix view object. The following instances are supported: vsip_mputattrib_f vsip_mputattrib_i vsip_mputattrib_si vsip_cmputattrib_f vsip_cmputattrib_i vsip_cmputattrib_si vsip_mputattrib_bl
vsip_Dmview_P * vsip_Dmputcollength_P(vsip_Dmview_P * matrix, vsip_length n2);	Set the column length attribute of a matrix view object. The following instances are supported: vsip_mputcollength_f vsip_mputcollength_i vsip_mputcollength_si vsip_cmputcollength_f vsip_cmputcollength_i vsip_cmputcollength_si vsip_mputcollength_bl
vsip_Dmview_P * vsip_Dmputcolstride_P(vsip_Dmview_P * matrix, vsip_stride s2);	Set the column stride attribute of a matrix view object. The following instances are supported: vsip_mputcolstride_f vsip_mputcolstride_i vsip_mputcolstride_si vsip_cmputcolstride_f vsip_cmputcolstride_i vsip_cmputcolstride_si vsip_mputcolstride_bl
vsip_Dmview_P * vsip_Dmputoffset_P(vsip_Dmview_P * matrix, vsip_offset offset);	Set the offset attribute of a matrix view object. The following instances are supported: vsip_mputoffset_f vsip_mputoffset_i vsip_mputoffset_si vsip_cmputoffset_f vsip_cmputoffset_i vsip_cmputoffset_si vsip_mputoffset_bl

Prototype	Description

`vsip_Dmview_P *`
`vsip_Dmputrowlength_P(`
`vsip_Dmview_P * matrix,`
`vsip_length n1);`
	Set the row length attribute of a matrix view object.
	The following instances are supported:
	`vsip_mputrowlength_f`
	`vsip_mputrowlength_i`
	`vsip_mputrowlength_si`
	`vsip_cmputrowlength_f`
	`vsip_cmputrowlength_i`
	`vsip_cmputrowlength_si`
	`vsip_mputrowlength_bl`

`vsip_Dmview_P *`
`vsip_Dmputrowstride_P(`
`vsip_Dmview_P * matrix,`
`vsip_stride s1);`
	Set the row stride attribute of a matrix view object.
	The following instances are supported:
	`vsip_mputrowstride_f`
	`vsip_mputrowstride_i`
	`vsip_mputrowstride_si`
	`vsip_cmputrowstride_f`
	`vsip_cmputrowstride_i`
	`vsip_cmputrowstride_si`
	`vsip_mputrowstride_bl`

`vsip_mview_P *`
`vsip_mrealview_P(`
`const vsip_cmview_P *`
`complex_matrix);`
	Create a matrix view object of the real part of complex matrix from a complex matrix view object.
	The following instances are supported:
	`vsip_mrealview_f`
	`vsip_mrealview_i`
	`vsip_mrealview_si`

`vsip_Dvview_P *`
`vsip_Dmrowview_P(`
`const vsip_Dmview_P * matrix,`
`vsip_index i);`
	Create a vector view object of a specified row of the source matrix view object.
	The following instances are supported:
	`vsip_mrowview_f`
	`vsip_mrowview_i`
	`vsip_mrowview_si`
	`vsip_cmrowview_f`
	`vsip_cmrowview_i`
	`vsip_cmrowview_si`
	`vsip_mrowview_bl`

`vsip_Dmview_P *`
`vsip_Dmsubview_P(`
`const vsip_Dmview_P * matrix,`
`vsip_index i,`
`vsip_index j,`
`vsip_length m,`
`vsip_length n);`
	Create a matrix view object that is a subview of matrix view object.
	The following instances are supported:
	`vsip_msubview_f`
	`vsip_msubview_i`
	`vsip_msubview_si`
	`vsip_cmsubview_f`
	`vsip_cmsubview_i`
	`vsip_cmsubview_si`
	`vsip_msubview_bl`

Prototype	Description
vsip_Dmview_P * vsip_Dmtransview_P(const vsip_Dmview_P * matrix);	Create a matrix view object that is the transpose of a matrix view object. The following instances are supported: vsip_mtransview_f vsip_mtransview_i vsip_mtransview_si vsip_cmtransview_f vsip_cmtransview_i vsip_cmtransview_si vsip_mtransview_bl

Chapter 2. Scalar Functions

2.1 Real Scalar Functions

Prototype	Description
`vsip_scalar_f vsip_acos_f( const vsip_scalar_f A);`	Computes the principal radian value in $[0, \pi]$ of the inverse cosine of a scalar.
`vsip_scalar_f vsip_asin_f( const vsip_scalar_f A);`	Computes the principal radian value in $[0, \pi]$ of the inverse sine of a scalar.
`vsip_scalar_f vsip_atan_f( const vsip_scalar_f A);`	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent of a scalar.
`vsip_scalar_f vsip_atan2_f( const vsip_scalar_f A, const vsip_scalar_f B);`	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of two scalars.
`vsip_scalar_f vsip_ceil_f( const vsip_scalar_f A);`	Computes the ceiling of a scalar.
`vsip_scalar_f vsip_cos_f( const vsip_scalar_f A);`	Computes the cosine of a scalar angle in radians.
`vsip_scalar_f vsip_cosh_f( const vsip_scalar_f A);`	Computes the hyperbolic cosine of a scalar.
`vsip_scalar_f vsip_exp_f( const vsip_scalar_f A);`	Computes the exponential of a scalar.
`vsip_scalar_f vsip_floor_f( const vsip_scalar_f A);`	Computes the floor of a scalar.
`vsip_scalar_f vsip_log_f( const vsip_scalar_f A);`	Computes the natural logarithm of a scalar.
`vsip_scalar_f vsip_log10_f( const vsip_scalar_f A);`	Computes the base 10 logarithm of a scalar.
`vsip_scalar_f vsip_mag_f( const vsip_scalar_f A);`	Computes the magnitude (absolute value) of a scalar.
`vsip_scalar_f vsip_pow_f( const vsip_scalar_f A, const vsip_scalar_f B);`	Computes the power function of two scalars.
`vsip_scalar_f vsip_sin_f( const vsip_scalar_f A);`	Computes the sine of a scalar angle in radians.
`vsip_scalar_f vsip_sinh_f( const vsip_scalar_f A);`	Computes the hyperbolic sine of a scalar.
`vsip_scalar_f vsip_sqrt_f( const vsip_scalar_f A);`	Computes the square root of a scalar.
`vsip_scalar_f vsip_tan_f( const vsip_scalar_f A);`	Computes the tangent of a scalar angle in radians.

Prototype	Description

`vsip_scalar_f`
`vsip_tanh_f(`
`const vsip_scalar_f A);`
	Computes the hyperbolic tangent of a scalar.

2.2 Complex Scalar Functions

Prototype	Description

`vsip_scalar_f`
`vsip_arg_f(`
`vsip_cscalar_f x);`
	Returns the argument in radians $[-\pi, \pi]$ of a complex scalar.

`void`
`vsip_CADD_f(`
`vsip_cscalar_f x,`
`vsip_cscalar_f y,`
`vsip_cscalar_f * z);`
	Computes the complex sum of two scalars.

`vsip_cscalar_f`
`vsip_cadd_f(`
`vsip_cscalar_f x,`
`vsip_cscalar_f y);`
	Computes the complex sum of two scalars.

`void`
`vsip_RCADD_f(`
`vsip_scalar_f x,`
`vsip_cscalar_f y,`
`vsip_cscalar_f * z);`
	Computes the complex sum of two scalars.

`vsip_cscalar_f`
`vsip_rcadd_f(`
`vsip_scalar_f x,`
`vsip_cscalar_f y);`
	Computes the complex sum of two scalars.

`void`
`vsip_CDIV_f(`
`vsip_cscalar_f x,`
`vsip_cscalar_f y,`
`vsip_cscalar_f * z);`
	Computes the complex quotient of two scalars.

`vsip_cscalar_f`
`vsip_cdiv_f(`
`vsip_cscalar_f x,`
`vsip_cscalar_f y);`
	Computes the complex quotient of two scalars.

`void`
`vsip_CRDIV_f(`
`vsip_cscalar_f x,`
`vsip_scalar_f y,`
`vsip_cscalar_f * z);`
	Computes the complex quotient of two scalars.

`vsip_cscalar_f`
`vsip_crdiv_f(`
`vsip_cscalar_f x,`
`vsip_scalar_f y);`
	Computes the complex quotient of two scalars.

`void`
`vsip_CEXP_f(`
`vsip_cscalar_f x,`
`vsip_cscalar_f * y);`
	Computes the exponential of a scalar.

`vsip_cscalar_f`
`vsip_cexp_f(`
`const vsip_cscalar_f A);`
	Computes the exponential of a scalar.

`void`
`vsip_CJMUL_f(`
`vsip_cscalar_f x,`
`vsip_cscalar_f y,`
`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.
`void vsip_RECT_f( vsip_scalar_f radius, vsip_scalar_f theta, vsip_cscalar_f * r);`	Convert a pair of real scalars from complex polar to complex rectangular form.
`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.

2.3 Index Scalar Functions

Prototype	Description

`void`
`vsip_MATINDEX(`
`vsip_index r,`
`vsip_index c,`
`vsip_scalar_mi * mi);`
	Form a matrix index from two vector indices.

`vsip_scalar_mi`
`vsip_matindex(`
`vsip_index r,`
`vsip_index c);`
	Form a matrix index from two vector indices.

`vsip_index`
`vsip_mcolindex(`
`vsip_scalar_mi mi);`
	Returns the column vector index from a matrix index.

`vsip_index`
`vsip_mrowindex(`
`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 * vsip_randcreate( const vsip_index seed, const vsip_index numprocs, const vsip_index id, const vsip_rng portable);`	Create a random number generator state object.
`int vsip_randdestroy( 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 vsip_randu_f( 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 vsip_crandu_f( 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 vsip_vrandu_f( vsip_randstate * state, 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 vsip_cvrandu_f( vsip_randstate * state, 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 vsip_randn_f( 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 vsip_crandn_f( 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 vsip_vacos_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a vector.
`void vsip_macos_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a matrix.
`void vsip_vasin_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a vector.
`void vsip_masin_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a matrix.
`void vsip_vatan_f( const vsip_vview_f * A, 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 vsip_matan_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a matrix.
`void vsip_vatan2_f( const vsip_vview_f * A, const vsip_vview_f * B, 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 vsip_matan2_f( const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);`	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input matrices.
`void vsip_vcos_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the cosine for each element of a vector. Element angle values are in radians.
`void vsip_mcos_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the cosine for each element of a matrix. Element angle values are in radians.
`void vsip_vcosh_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the hyperbolic cosine for each element of a vector.
`void vsip_mcosh_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the hyperbolic cosine for each element of a matrix.
`void vsip_vexp_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the exponential function value for each element of a vector.

Prototype	Description
`void vsip_cvexp_f( const vsip_cvview_f * A, const vsip_cvview_f * R);`	Computes the exponential function value for each element of a vector.
`void vsip_mexp_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the exponential function value for each element of a matrix.
`void vsip_cmexp_f( const vsip_cmview_f * A, const vsip_cmview_f * R);`	Computes the exponential function value for each element of a matrix.
`void vsip_vexp10_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the base 10 exponential for each element of a vector.
`void vsip_mexp10_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the base 10 exponential for each element of a matrix.
`void vsip_vlog_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the natural logarithm for each element of a vector.
`void vsip_cvlog_f( const vsip_cvview_f * A, const vsip_cvview_f * R);`	Computes the natural logarithm for each element of a vector.
`void vsip_mlog_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the natural logarithm for each element of a matrix.
`void vsip_cmlog_f( const vsip_cmview_f * A, const vsip_cmview_f * R);`	Computes the natural logarithm for each element of a matrix.
`void vsip_vlog10_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Compute the base ten logarithm for each element of a vector.
`void vsip_mlog10_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Compute the base ten logarithm for each element of a matrix.
`void vsip_vsin_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Compute the sine for each element of a vector. Element angle values are in radians.
`void vsip_msin_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Compute the sine for each element of a matrix. Element angle values are in radians.
`void vsip_vsinh_f( const vsip_vview_f * A, const vsip_vview_f * R);`	Computes the hyperbolic sine for each element of a vector.
`void vsip_msinh_f( const vsip_mview_f * A, const vsip_mview_f * R);`	Computes the hyperbolic sine for each element of a matrix.

Prototype	Description
void vsip_Dvsqrt_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);	Compute the square root for each element of a vector. The following instances are supported: vsip_vsqrt_f vsip_cvsqrt_f
void vsip_Dmsqrt_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);	Compute the square root for each element of a matrix. The following instances are supported: vsip_msqrt_f vsip_cmsqrt_f
void vsip_vtan_f(const vsip_vview_f * A, const vsip_vview_f * R);	Compute the tangent for each element of a vector. Element angle values are in radians.
void vsip_mtan_f(const vsip_mview_f * A, const vsip_mview_f * R);	Compute the tangent for each element of a matrix. Element angle values are in radians.
void vsip_vtanh_f(const vsip_vview_f * A, const vsip_vview_f * R);	Computes the hyperbolic tangent for each element of a vector.
void vsip_mtanh_f(const vsip_mview_f * A, const vsip_mview_f * R);	Computes the hyperbolic tangent for each element of a matrix.

4.2 Unary Operations

Prototype	Description
void vsip_varg_f(const vsip_cvview_f * A, const vsip_vview_f * R);	Computes the argument in radians $[-\pi, \pi]$ for each element of a complex vector.
void vsip_marg_f(const vsip_cmview_f * A, const vsip_mview_f * R);	Computes the argument in radians $[-\pi, \pi]$ for each element of a complex matrix.
void vsip_vceil_f(const vsip_vview_f * A, const vsip_vview_f * R);	Computes the ceiling for each element of a vector.
void vsip_cvconj_f(const vsip_cvview_f * A, const vsip_cvview_f * R);	Compute the conjugate for each element of a complex vector.
void vsip_cmconj_f(const vsip_cmview_f * A, const vsip_cmview_f * R);	Compute the conjugate for each element of a complex matrix.

Prototype	Description
void vsip_Dvcumsum_P(const vsip_Dvview_f * A, const vsip_Dvview_f * R);	Compute the cumulative sum of the elements of a vector. The following instances are supported: vsip_vcumsum_f vsip_vcumsum_i vsip_cvcumsum_f vsip_cvcumsum_i
void vsip_Dmcumsum_P(vsip_major dir, const vsip_Dmview_f * R);	Compute the cumulative sums of the elements in the rows or columns of a matrix. The following instances are supported: vsip_mcumsum_f vsip_mcumsum_i vsip_cmcumsum_f vsip_cmcumsum_i
void vsip_veuler_f(const vsip_vview_f * A, 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 vsip_meuler_f(const vsip_mview_f * A, const vsip_cmview_f * R);	Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a matrix.
void vsip_vfloor_f(const vsip_vview_f * A, const vsip_vview_f * R);	Computes the floor for each element of a vector.
void vsip_Dvmag_P(const vsip_Dvview_P * A, const vsip_vview_P * R);	Compute the magnitude for each element of a vector. The following instances are supported: vsip_vmag_f vsip_vmag_i vsip_vmag_si vsip_cvmag_f
void vsip_Dmmag_P(const vsip_Dmview_P * A, const vsip_mview_P * R);	Compute the magnitude for each element of a matrix. The following instances are supported: vsip_mmag_f vsip_cmmag_f
void vsip_vcmagsq_f(const vsip_cvview_f * A, const vsip_vview_f * R);	Computes the square of the magnitudes for each element of a vector.
void vsip_mcmagsq_f(const vsip_cmview_f * A, const vsip_mview_f * R);	Computes the square of the magnitudes for each element of a matrix.
vsip_Dscalar_P vsip_Dvmeanval_P(const vsip_Dvview_P * A);	Returns the mean value of the elements of a vector. The following instances are supported: vsip_vmeanval_f vsip_cvmeanval_f

Prototype	Description
vsip_Dscalar_P vsip_Dmmeanval_P(const vsip_Dmview_P * A);	Returns the mean value of the elements of a matrix. The following instances are supported: vsip_mmeanval_f vsip_cmmeanval_f
vsip_scalar_P vsip_Dvmeansqval_P(const vsip_Dvview_P * A);	Returns the mean magnitude squared value of the elements of a vector. The following instances are supported: vsip_vmeansqval_f vsip_cvmeansqval_f
vsip_scalar_P vsip_Dmmeansqval_P(const vsip_Dmview_P * A);	Returns the mean magnitude squared value of the elements of a matrix. The following instances are supported: vsip_mmeansqval_f vsip_cmmeansqval_f
vsip_scalar_P vsip_Dvmodulate_P(const vsip_Dvview_P * A, const vsip_scalar_P nu, const vsip_scalar_P phi, const vsip_Dvview_P * R);	Computes the modulation of a real vector by a specified complex frequency. The following instances are supported: vsip_vmodulate_f vsip_cvmodulate_f
void vsip_Dvneg_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);	Computes the negation for each element of a vector. The following instances are supported: vsip_vneg_f vsip_vneg_i vsip_vneg_si vsip_cvneg_f
void vsip_Dmneg_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);	Computes the negation for each element of a matrix. The following instances are supported: vsip_mneg_f vsip_mneg_i vsip_cmneg_f
void vsip_Dvrecip_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);	Computes the reciprocal for each element of a vector. The following instances are supported: vsip_vrecip_f vsip_cvrecip_f
void vsip_Dmrecip_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);	Computes the reciprocal for each element of a matrix. The following instances are supported: vsip_mrecip_f vsip_cmrecip_f
void vsip_vrsqrt_f(const vsip_vview_f * A, const vsip_vview_f * R);	Computes the reciprocal of the square root for each element of a vector.

Prototype	Description
void vsip_mrsqrt_f(const vsip_mview_f * A, const vsip_mview_f * R);	Computes the reciprocal of the square root for each element of a matrix.
void vsip_vsq_f(const vsip_vview_f * A, const vsip_vview_f * R);	Computes the square for each element of a vector.
void vsip_msq_f(const vsip_mview_f * A, const vsip_mview_f * R);	Computes the square for each element of a matrix.
vsip_scalar_P vsip_Dvsumval_P(const vsip_vview_P * A);	Returns the sum of the elements of a vector. The following instances are supported: vsip_vsumval_f vsip_vsumval_i vsip_vsumval_si vsip_cvsumval_f vsip_cvsumval_i
vsip_scalar_P vsip_Dmsumval_P(const vsip_mview_P * A);	Returns the sum of the elements of a vector. The following instances are supported: vsip_msumval_f vsip_msumval_i vsip_cmsumval_f vsip_cmsumval_i
vsip_scalar_bl vsip_vsumval_bl(const vsip_vview_bl * A);	Returns the sum of the elements of a vector.
vsip_scalar_f vsip_vsumsqval_f(const vsip_vview_f * A);	Returns the sum of the squares of the elements of a vector.
vsip_scalar_f vsip_msumsqval_f(const vsip_mview_f * A);	Returns the sum of the squares of the elements of a matrix.

4.3 Binary Operations

Prototype	Description
void vsip_Dvadd_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the sum, by element, of two vectors. The following instances are supported: vsip_vadd_f vsip_vadd_i vsip_vadd_si vsip_cvadd_f vsip_cvadd_i
void vsip_Dmadd_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the sum, by element, of two matrices. The following instances are supported: vsip_madd_f vsip_madd_i vsip_cmadd_f

Prototype	Description
void vsip_rcvadd_f(const vsip_vview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the sum, by element, of two vectors.
void vsip_rcmadd_f(const vsip_mview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the sum, by element, of two matrices.
void vsip_Dsvadd_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the sum, by element, of a scalar and a vector. The following instances are supported: vsip_svadd_f vsip_svadd_i vsip_svadd_si vsip_csvadd_f
void vsip_Dsmadd_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the sum, by element, of a scalar and a matrix. The following instances are supported: vsip_smadd_f vsip_smadd_i vsip_csmadd_f
void vsip_rscvadd_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the sum, by element, of a real scalar and a complex vector.
void vsip_rscmadd_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the sum, by element, of a real scalar and a complex matrix.
void vsip_Dvdiv_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the quotient, by element, of two vectors. The following instances are supported: vsip_vdiv_f vsip_cvdiv_f
void vsip_Dmdiv_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the quotient, by element, of two matrices. The following instances are supported: vsip_mdiv_f vsip_cmdiv_f
void vsip_rcvdiv_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the quotient, by element, of two vectors.
void vsip_rcmdiv_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the quotient, by element, of two matrices.

Prototype	Description
void vsip_crvdiv_f(const vsip_cvview_f * A, const vsip_vview_f * B, const vsip_cvview_f * R);	Computes the quotient, by element, of two vectors.
void vsip_crmdiv_f(const vsip_cmview_f * A, const vsip_mview_f * B, const vsip_cmview_f * R);	Computes the quotient, by element, of two matrices.
void vsip_svdiv_f(const vsip_scalar_f a, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the quotient, by element, of a scalar and a vector.
void vsip_Dsmdiv_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the quotient, by element, of a scalar and a matrix. The following instances are supported: vsip_smdiv_f vsip_csmdiv_f
void vsip_rscvdiv_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the quotient, by element, of a real scalar and a complex vector.
void vsip_rscvsub_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the difference, by element, of a real scalar and a complex vector.
void vsip_rscmdiv_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the quotient, by element, of a real scalar and a complex matrix.
void vsip_rscmsub_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the difference, by element, of a real scalar and a complex matrix.
void vsip_Dvsdiv_P(const vsip_Dvview_P * A, const vsip_Dscalar_P b, const vsip_Dvview_P * R);	Computes the quotient, by element, of a vector and a scalar. The following instances are supported: vsip_vsdiv_f vsip_cvrsdiv_f
void vsip_Dmsdiv_P(const vsip_Dmview_P * A, const vsip_Dscalar_P b, const vsip_Dmview_P * R);	Computes the quotient, by element, of a matrix and a scalar. The following instances are supported: vsip_msdiv_f vsip_cmrsdiv_f
void vsip_Dvexpoavg_P(const vsip_scalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * C);	Computes an exponential weighted average, by element, of two vectors. The following instances are supported: vsip_vexpoavg_f vsip_cvexpoavg_f

Prototype	Description
void vsip_Dmexpoavg_P(const vsip_scalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * C);	Computes an exponential weighted average, by element, of two matrices. The following instances are supported: vsip_mexpoavg_f vsip_cmexpoavg_f
void vsip_vhypot_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the square root of the sum of squares, by element, of two input vectors.
void vsip_mhypot_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);	Computes the square root of the sum of squares, by element, of two input matrices.
void vsip_cvjmul_f(const vsip_cvview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the product of a complex vector with the conjugate of a second complex vector, by element.
void vsip_cmjmul_f(const vsip_cmview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.
void vsip_Dvmul_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the product, by element, of two vectors. The following instances are supported: vsip_vmul_f vsip_vmul_i vsip_vmul_si vsip_cvmul_f
void vsip_Dmmul_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the product, by element, of two matrices. The following instances are supported: vsip_mmul_f vsip_mmul_i vsip_cmmul_f
void vsip_rcvmul_f(const vsip_vview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the product, by element, of two vectors.
void vsip_rcmmul_f(const vsip_mview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the product, by element, of two matrices.
void vsip_Dsvmul_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the product, by element, of a scalar and a vector. The following instances are supported: vsip_svmul_f vsip_svmul_i vsip_svmul_si vsip_csvmul_f

Prototype	Description
void vsip_Dsmmul_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the product, by element, of a scalar and a matrix. The following instances are supported: vsip_smmul_f vsip_csmmul_f
void vsip_rscvmul_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the product, by element, of a real scalar and a complex vector.
void vsip_rscmmul_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the product, by element, of a real scalar and a complex matrix.
void vsip_DvDmmul_P(const vsip_Dvview_P * A, const vsip_Dmview_P * B, const vsip_major major, const vsip_Dmview_P * R);	Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported: vsip_vmmul_f vsip_cvmmul_f
void vsip_rvcmmul_f(const vsip_vview_f * A, const vsip_cmview_f * B, const vsip_major major, const vsip_cmview_f * R);	Computes the product, by element, of a vector and the rows or columns of a matrix.
void vsip_Dvsub_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the difference, by element, of two vectors. The following instances are supported: vsip_vsub_f vsip_vsub_i vsip_vsub_si vsip_cvsub_f
void vsip_Dmsub_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the difference, by element, of two matrices. The following instances are supported: vsip_msub_f vsip_msub_i vsip_cmsub_f
void vsip_crvsub_f(const vsip_cvview_f * A, const vsip_vview_f * B, const vsip_cvview_f * R);	Computes the difference, by element, of two vectors.
void vsip_crmsub_f(const vsip_cmview_f * A, const vsip_mview_f * B, const vsip_cmview_f * R);	Computes the difference, by element, of two matrices.
void vsip_rcvsub_f(const vsip_vview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);	Computes the difference, by element, of two vectors.

Prototype	Description
void vsip_rcmsub_f(const vsip_mview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);	Computes the difference, by element, of two matrices.
void vsip_Dsvsub_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);	Computes the difference, by element, of a scalar and a vector. The following instances are supported: vsip_svsub_f vsip_svsub_i vsip_svsub_si vsip_csvsub_f
void vsip_Dsmsub_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Computes the difference, by element, of a scalar and a matrix. The following instances are supported: vsip_smsub_f vsip_smsub_i vsip_csmsub_f

4.4 Ternary Operations

Prototype	Description
void vsip_Dvam_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * C, const vsip_Dvview_P * R);	Computes the sum of two vectors and product of a third vector, by element. The following instances are supported: vsip_vam_f vsip_cvam_f
void vsip_Dvma_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * C, const vsip_Dvview_P * R);	Computes the product of two vectors and sum of a third vector, by element. The following instances are supported: vsip_vma_f vsip_cvma_f
void vsip_Dvmsa_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dscalar_P c, const vsip_Dvview_P * R);	Computes the product of two vectors and sum of a scalar, by element. The following instances are supported: vsip_vmsa_f vsip_cvmsa_f
void vsip_Dvmsb_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * C, const vsip_Dvview_P * R);	Computes the product of two vectors and difference of a third vector, by element. The following instances are supported: vsip_vmsb_f vsip_cvmsb_f
void vsip_Dvsam_P(const vsip_Dvview_P * A, const vsip_Dscalar_P b, const vsip_Dvview_P * C, const vsip_Dvview_P * R);	Computes the sum of a vector and a scalar, and product with a second vector, by element. The following instances are supported: vsip_vsam_f vsip_cvsam_f

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

4.5 Logical Operations

Prototype	Description
vsip_scalar_bl vsip_valltrue_bl(const vsip_vview_bl * A);	Returns true if all the elements of a vector are true.
vsip_scalar_bl vsip_malltrue_bl(const vsip_vview_bl * A);	Returns true if all the elements of a vector are true.
vsip_scalar_bl vsip_vanytrue_bl(const vsip_vview_bl * A);	Returns true if one or more elements of a vector are true.
vsip_scalar_bl vsip_manytrue_bl(const vsip_vview_bl * A);	Returns true if one or more elements of a vector are true.
void vsip_Dvleq_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_vview_bl * R);	Computes the boolean comparison of 'equal', by element, of two vectors. The following instances are supported: vsip_vleq_f vsip_vleq_i vsip_cvleq_f vsip_cvleq_i
void vsip_Dmleq_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_vview_bl * R);	Computes the boolean comparison of 'equal', by element, of two vectors/matrices. The following instances are supported: vsip_mleq_f vsip_mleq_i vsip_cmleq_f vsip_cmleq_i

Prototype	Description

`void`
`vsip_vlge_f(`
`const vsip_vview_f * A,`
`const vsip_vview_f * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'greater than or equal', by element, of two vectors.

`void`
`vsip_mlge_P(`
`const vsip_mview_P * A,`
`const vsip_mview_P * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'greater than or equal', by element, of two vectors/matrices.
	The following instances are supported:
	`vsip_mlge_f`
	`vsip_mlge_i`

`void`
`vsip_vlgt_f(`
`const vsip_vview_f * A,`
`const vsip_vview_f * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'greater than', by element, of two vectors.

`void`
`vsip_mlgt_P(`
`const vsip_mview_P * A,`
`const vsip_mview_P * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'greater than', by element, of two vectors/matrices.
	The following instances are supported:
	`vsip_mlgt_f`
	`vsip_mlgt_i`

`void`
`vsip_vlle_f(`
`const vsip_vview_f * A,`
`const vsip_vview_f * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'less than or equal', by element, of two vectors.

`void`
`vsip_mlle_P(`
`const vsip_mview_P * A,`
`const vsip_mview_P * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'less than or equal', by element, of two vectors/matrices.
	The following instances are supported:
	`vsip_mlle_f`
	`vsip_mlle_i`

`void`
`vsip_vllt_f(`
`const vsip_vview_f * A,`
`const vsip_vview_f * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'less than', by element, of two vectors.

`void`
`vsip_mllt_P(`
`const vsip_mview_P * A,`
`const vsip_mview_P * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'less than', by element, of two vectors/matrices.
	The following instances are supported:
	`vsip_mllt_f`
	`vsip_mllt_i`

`void`
`vsip_Dvlne_P(`
`const vsip_Dvview_P * A,`
`const vsip_Dvview_P * B,`
`const vsip_vview_bl * R);`
	Computes the boolean comparison of 'not equal', by element, of two vectors.
	The following instances are supported:
	`vsip_vlne_f`
	`vsip_cvlne_f`
	`vsip_cvlne_i`

Prototype	Description
void vsip_Dmlne_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_vview_bl * R);	Computes the boolean comparison of 'not equal', by element, of two vectors/matrices. The following instances are supported: vsip_mlne_f vsip_mlne_i vsip_cmlne_f vsip_cmlne_i

4.6 Selection Operations

Prototype	Description
void vsip_vclip_P(const vsip_vview_P * A, const vsip_scalar_P t1, const vsip_scalar_P t2, const vsip_scalar_P c1, const vsip_scalar_P c2, const vsip_vview_P * R);	Computes the generalised double clip, by element, of two vectors. The following instances are supported: vsip_vclip_f vsip_vclip_i vsip_vclip_si
void vsip_vinvclip_P(const vsip_vview_P * A, const vsip_scalar_P t1, const vsip_scalar_P t2, const vsip_scalar_P t3, const vsip_scalar_P c1, const vsip_scalar_P c2, const vsip_vview_P * R);	Computes the generalised inverted double clip, by element, of two vectors. The following instances are supported: vsip_vinvclip_f vsip_vinvclip_i vsip_vinvclip_si
vsip_length vsip_vindexbool(const vsip_vview_bl * X, 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 vsip_vmax_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the maximum, by element, of two vectors.
void vsip_vmaxmg_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the maximum magnitude (absolute value), by element, of two vectors.
void vsip_vcmaxmgsq_f(const vsip_cvview_f * A, const vsip_cvview_f * B, const vsip_vview_f * R);	Computes the maximum magnitude squared, by element, of two complex vectors.
vsip_scalar_f vsip_vcmaxmgsqval_f(const vsip_cvview_f * A, 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 vsip_vmaxmgval_f(const vsip_vview_f * A, 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.

Prototype	Description
vsip_scalar_f vsip_vmaxval_f(const vsip_vview_f * A, 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 vsip_vmin_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the minimum, by element, of two vectors.
void vsip_vminmg_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);	Computes the minimum magnitude (absolute value), by element, of two vectors.
void vsip_vcminmgsq_f(const vsip_cvview_f * A, const vsip_cvview_f * B, const vsip_vview_f * R);	Computes the minimum magnitude squared, by element, of two complex vectors.
vsip_scalar_f vsip_vcminmgsqval_f(const vsip_cvview_f * A, 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 vsip_vminmgval_f(const vsip_vview_f * A, 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 vsip_vminval_f(const vsip_vview_f * A, 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 vsip_vand_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_P * R);	Computes the bitwise and, by element, of two vectors. The following instances are supported: vsip_vand_i vsip_vand_si
void vsip_mand_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_mview_P * R);	Computes the bitwise and, by element, of two matrices. The following instances are supported: vsip_mand_i vsip_mand_si
void vsip_vand_bl(const vsip_vview_bl * A, const vsip_vview_bl * B, const vsip_vview_bl * R);	Computes the bitwise and, by element, of two vectors.
void vsip_mand_bl(const vsip_mview_bl * A, const vsip_mview_bl * B, const vsip_mview_bl * R);	Computes the bitwise and, by element, of two matrices.

Prototype	Description
void vsip_vnot_P(const vsip_vview_P * A, const vsip_vview_P * R);	Computes the bitwise not (one's complement), by element, of two vectors. The following instances are supported: vsip_vnot_i vsip_vnot_si
void vsip_mnot_P(const vsip_mview_P * A, const vsip_mview_P * R);	Computes the bitwise not (one's complement), by element, of two matrices. The following instances are supported: vsip_mnot_i vsip_mnot_si
void vsip_vnot_bl(const vsip_vview_bl * A, const vsip_vview_bl * R);	Computes the bitwise not (one's complement), by element, of two vectors.
void vsip_mnot_bl(const vsip_mview_bl * A, const vsip_mview_bl * R);	Computes the bitwise not (one's complement), by element, of two matrices.
void vsip_vor_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_P * R);	Computes the bitwise inclusive or, by element, of two vectors. The following instances are supported: vsip_vor_i vsip_vor_si
void vsip_mor_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_mview_P * R);	Computes the bitwise inclusive or, by element, of two matrices. The following instances are supported: vsip_mor_i vsip_mor_si
void vsip_vor_bl(const vsip_vview_bl * A, const vsip_vview_bl * B, const vsip_vview_bl * R);	Computes the bitwise inclusive or, by element, of two vectors.
void vsip_mor_bl(const vsip_mview_bl * A, const vsip_mview_bl * B, const vsip_mview_bl * R);	Computes the bitwise inclusive or, by element, of two matrices.
void vsip_vxor_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_P * R);	Computes the bitwise exclusive or, by element, of two vectors. The following instances are supported: vsip_vxor_i vsip_vxor_si
void vsip_mxor_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_mview_P * R);	Computes the bitwise exclusive or, by element, of two matrices. The following instances are supported: vsip_mxor_i vsip_mxor_si

Prototype	Description
void vsip_vxor_bl(const vsip_vview_bl * A, const vsip_vview_bl * B, const vsip_vview_bl * R);	Computes the bitwise exclusive or, by element, of two vectors.
void vsip_mxor_bl(const vsip_mview_bl * A, const vsip_mview_bl * B, const vsip_mview_bl * R);	Computes the bitwise exclusive or, by element, of two matrices.

4.8 Element Generation and Copy

Prototype	Description
void vsip_Dvcopy_P_P(const vsip_Dvview_P * A, 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. The following instances are supported: vsip_vcopy_f_f vsip_vcopy_f_i vsip_vcopy_f_si vsip_vcopy_f_bl vsip_vcopy_i_f vsip_vcopy_i_i vsip_vcopy_i_si vsip_vcopy_i_vi vsip_vcopy_si_f vsip_vcopy_si_i vsip_vcopy_si_si vsip_vcopy_bl_f vsip_vcopy_bl_bl vsip_vcopy_vi_i vsip_vcopy_vi_vi vsip_vcopy_mi_mi vsip_cvcopy_f_f
void vsip_Dmcopy_P_P(const vsip_Dmview_P * A, 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. The following instances are supported: vsip_mcopy_f_f vsip_mcopy_f_i vsip_mcopy_f_si vsip_mcopy_f_bl vsip_mcopy_i_f vsip_mcopy_i_i vsip_mcopy_i_si vsip_mcopy_si_f vsip_mcopy_si_i vsip_mcopy_si_si vsip_mcopy_bl_f vsip_mcopy_bl_bl vsip_cmcopy_f_f

Prototype	Description
void vsip_Dvfill_P(const vsip_Dscalar_P a, const vsip_Dvview_P * R);	Fill a vector with a constant value. The following instances are supported: vsip_vfill_f vsip_vfill_i vsip_vfill_si vsip_cvfill_f
void vsip_Dmfill_P(const vsip_Dscalar_P a, const vsip_Dmview_P * R);	Fill a matrix with a constant value. The following instances are supported: vsip_mfill_f vsip_mfill_i vsip_mfill_si vsip_cmfill_f
void vsip_vramp_P(const vsip_scalar_P alpha, const vsip_scalar_P beta, 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. The following instances are supported: vsip_vramp_f vsip_vramp_i vsip_vramp_si

4.9 Manipulation Operations

Prototype	Description
void vsip_vcmplx_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_cvview_f * R);	Form a complex vector from two real vectors.
void vsip_mcmplx_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_cmview_f * R);	Form a complex matrix from two real matrices.
void vsip_Dvgather_P(const vsip_Dvview_P * X, const vsip_vview_vi * I, 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. The following instances are supported: vsip_vgather_f vsip_vgather_i vsip_vgather_si vsip_cvgather_f

Prototype	Description
void vsip_Dmgather_P(const vsip_Dmview_P * X, const vsip_vview_mi * I, const vsip_Dvview_f * Y);	The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same. The following instances are supported: vsip_mgather_f vsip_mgather_i vsip_mgather_si vsip_cmgather_f
void vsip_vimag_f(const vsip_cvview_f * A, const vsip_vview_f * R);	Extract the imaginary part of a complex vector.
void vsip_mimag_f(const vsip_cmview_f * A, const vsip_mview_f * R);	Extract the imaginary part of a complex matrix.
void vsip_vpolar_f(const vsip_cvview_f * A, const vsip_vview_f * R, 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 vsip_mpolar_f(const vsip_cmview_f * A, const vsip_mview_f * R, const vsip_mview_f * P);	Convert a complex matrix from rectangular to polar form. The polar data consists of a real matrix containing the radius and a corresponding real matrix containing the argument (angle) of the complex input data.
void vsip_vreal_f(const vsip_cvview_f * A, const vsip_vview_f * R);	Extract the real part of a complex vector.
void vsip_mreal_f(const vsip_cmview_f * A, const vsip_mview_f * R);	Extract the real part of a complex matrix.
void vsip_vrect_f(const vsip_vview_f * R, const vsip_vview_f * P, const vsip_cvview_f * A);	Convert a pair of real vectors from complex polar to complex rectangular form.
void vsip_mrect_f(const vsip_mview_f * R, const vsip_mview_f * P, const vsip_cmview_f * A);	Convert a pair of real matrices from complex polar to complex rectangular form.

Prototype	Description
void vsip_Dvscatter_P(const vsip_Dvview_P * X, const vsip_Dvview_P * Y, 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. The following instances are supported: vsip_vscatter_f vsip_vscatter_i vsip_vscatter_si vsip_cvscatter_f
void vsip_Dmscatter_P(const vsip_Dvview_f * X, const vsip_Dmview_f * Y, const vsip_vview_mi * I);	The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector. The following instances are supported: vsip_mscatter_f vsip_mscatter_i vsip_mscatter_si vsip_cmscatter_f
void vsip_Dvswap_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B);	Swap elements between two vectors. The following instances are supported: vsip_vswap_f vsip_cvswap_f
void vsip_Dmswap_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B);	Swap elements between two matrices. The following instances are supported: vsip_mswap_f vsip_mswap_i vsip_mswap_si vsip_cmswap_f

4.10 Extensions

Prototype	Description
vsip_scalar_f vsip_vcsummgval_f(const vsip_cvview_f * A);	Returns the sum of the magnitudes of the elements of a complex vector.

Chapter 5. Signal Processing Functions

5.1 FFT Functions

Prototype	Description

`vsip_fft_f *`
`vsip_ccfftip_create_f(`
`const vsip_index length,`
`const vsip_scalar_f scale,`
`const vsip_fft_dir dir,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a 1D FFT object.

`vsip_fft_f *`
`vsip_ccfftop_create_f(`
`const vsip_index length,`
`const vsip_scalar_f scale,`
`const vsip_fft_dir dir,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a 1D FFT object.

`vsip_fft_f *`
`vsip_crfftop_create_f(`
`const vsip_index length,`
`const vsip_scalar_f scale,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a 1D FFT object.

`vsip_fft_f *`
`vsip_rcfftop_create_f(`
`const vsip_index length,`
`const vsip_scalar_f scale,`
`const vsip_length ntimes,`
`const vsip_alg_hint hint);`
	Create a 1D FFT object.

`int`
`vsip_fft_destroy_f(`
`vsip_fft_f * plan);`
	Destroy an FFT object.

`void`
`vsip_fft_getattr_f(`
`const vsip_fft_f * plan,`
`vsip_fft_attr_f * attr);`
	Return the attributes of an FFT object.

`void`
`vsip_ccfftip_f(`
`const vsip_fft_f * plan,`
`const vsip_cvview_f * xy);`
	Apply a complex-to-complex Fast Fourier Transform (FFT).

`void`
`vsip_ccfftop_f(`
`const vsip_fft_f * plan,`
`const vsip_cvview_f * x,`
`const vsip_cvview_f * y);`
	Apply a complex-to-complex Fast Fourier Transform (FFT).

`void`
`vsip_crfftop_f(`
`const vsip_fft_f * plan,`
`const vsip_cvview_f * x,`
`const vsip_vview_f * y);`
	Apply a complex-to-real Fast Fourier Transform (FFT).

`void`
`vsip_rcfftop_f(`
`const vsip_fft_f * plan,`
`const vsip_vview_f * x,`
`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).

Prototype	Description
vsip_fft2d_f * vsip_ccfft2dip_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_length ntimes, const vsip_alg_hint hint);	Create a 2D FFT object.
vsip_fft2d_f * vsip_ccfft2dop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_length ntimes, const vsip_alg_hint hint);	Create a 2D FFT object.
vsip_fft2d_f * vsip_crfft2dop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_length ntimes, const vsip_alg_hint hint);	Create a 2D FFT object.
vsip_fft2d_f * vsip_rcfft2dop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_length ntimes, const vsip_alg_hint hint);	Create a 2D FFT object.
int vsip_fft2d_destroy_f(vsip_fft2d_f * plan);	Destroy an FFT object.
void vsip_fft2d_getattr_f(const vsip_fft2d_f * plan, vsip_fft2d_attr_f * attr);	Return the attributes of an FFT object.
void vsip_ccfft2dip_f(const vsip_fft2d_f * plan, const vsip_cmview_f * XY);	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).
void vsip_ccfft2dop_f(const vsip_fft2d_f * plan, const vsip_cmview_f * X, const vsip_cmview_f * Y);	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).
void vsip_crfft2dop_f(const vsip_fft2d_f * plan, const vsip_cmview_f * X, const vsip_mview_f * Y);	Apply a complex-to-real 2D Fast Fourier Transform (FFT).
void vsip_rcfft2dop_f(const vsip_fft2d_f * plan, const vsip_mview_f * X, const vsip_cmview_f * Y);	Apply a real-to-complex 2D Fast Fourier Transform (FFT).

5.2 Convolution/Correlation Functions

Prototype	Description
vsip_conv1d_f * vsip_conv1d_create_f(const vsip_vview_f * kernel, const vsip_symmetry symm, const vsip_length N, const vsip_length D, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);	Create a decimated 1D convolution filter object.
int vsip_conv1d_destroy_f(vsip_conv1d_f * plan);	Destroy a 1D convolution object.
void vsip_conv1d_getattr_f(const vsip_conv1d_f * plan, vsip_conv1d_attr_f * attr);	Returns the attributes for a 1D convolution object.
void vsip_convolve1d_f(const vsip_conv1d_f * plan, const vsip_vview_f * x, const vsip_vview_f * y);	Compute a decimated real one-dimensional (1D) convolution of two vectors.
vsip_conv2d_f * vsip_conv2d_create_f(const vsip_mview_f * H, const vsip_symmetry symm, const vsip_length P, const vsip_length Q, const vsip_length D, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);	Create a decimated 2D convolution filter object.
int vsip_conv2d_destroy_f(vsip_conv2d_f * plan);	Destroy a 2D convolution object.
void vsip_conv2d_getattr_f(const vsip_conv2d_f * plan, vsip_conv2d_attr_f * attr);	Returns the attributes for a 2D convolution object.
void vsip_convolve2d_f(const vsip_conv2d_f * plan, const vsip_mview_f * x, const vsip_mview_f * y);	Compute a decimated real two-dimensional (2D) convolution of two matrices.
vsip_Dcorr1d_P * vsip_Dcorr1d_create_P(const vsip_length M, const vsip_length N, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);	Create a 1D correlation object. The following instances are supported: vsip_corr1d_create_f vsip_ccorr1d_create_f
int vsip_Dcorr1d_destroy_P(vsip_Dcorr1d_P * plan);	Destroy a 1D correlation object. The following instances are supported: vsip_corr1d_destroy_f vsip_ccorr1d_destroy_f

Prototype	Description
void vsip_Dcorr1d_getattr_P(const vsip_Dcorr1d_P * plan, vsip_Dcorr1d_attr_P * attr);	Return the attributes for a 1D correlation object. The following instances are supported: vsip_corr1d_getattr_f vsip_ccorr1d_getattr_f
void vsip_Dcorrelate1d_P(const vsip_Dcorr1d_P * plan, const vsip_bias bias, const vsip_Dvview_P * ref, const vsip_Dvview_P * x, const vsip_Dvview_P * y);	Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported: vsip_correlate1d_f vsip_ccorrelate1d_f
*vsip_Dcorr2d_P ** vsip_Dcorr2d_create_P(const vsip_length M, const vsip_length N, vsip_length P, vsip_length Q, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);	Create a 2D correlation object. The following instances are supported: vsip_corr2d_create_f vsip_ccorr2d_create_f
int vsip_Dcorr2d_destroy_P(vsip_Dcorr2d_P * plan);	Destroy a 2D correlation object. The following instances are supported: vsip_corr2d_destroy_f vsip_ccorr2d_destroy_f
void vsip_Dcorr2d_getattr_P(const vsip_Dcorr12_P * plan, vsip_Dcorr12_attr_P * attr);	Return the attributes for a 2D correlation object. The following instances are supported: vsip_corr2d_getattr_f vsip_ccorr2d_getattr_f
void vsip_Dcorrelate2d_P(const vsip_Dcorr2d_P * plan, const vsip_bias bias, const vsip_Dmview_P * ref, const vsip_Dmview_P * x, const vsip_Dmview_P * y);	Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported: vsip_correlate2d_f vsip_ccorrelate2d_f

5.3 Window Functions

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

Prototype	Description

`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`

`void`
`vsip_Dfir_reset_P(`
`vsip_Dfir_P * fir);`
	Reset the state of a decimated FIR filter object.
	The following instances are supported:
	`vsip_fir_reset_f`
	`vsip_cfir_reset_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`
`vsip_cmherm_f(`
`const vsip_cmview_f * A,`
`const vsip_cmview_f * R);`
	Complex Hermitian (conjugate transpose) of a matrix.

`vsip_cscalar_f`
`vsip_cvjdot_f(`
`const vsip_cvview_f * A,`
`const vsip_cvview_f * B);`
	Compute the conjugate inner (dot) product of two complex vectors.

`void`
`vsip_gemp_f(`
`const vsip_scalar_f alpha,`
`const vsip_mview_f * A,`
`const vsip_mat_op Aop,`
`const vsip_mview_f * B,`
`const vsip_mat_op Bop,`
`const vsip_scalar_f beta,`
`const vsip_mview_f * R);`
	Calculate the general product of two matrices and accumulate.

`void`
`vsip_cgemp_f(`
`const vsip_cscalar_f alpha,`
`const vsip_cmview_f * A,`
`const vsip_mat_op Aop,`
`const vsip_cmview_f * B,`
`const vsip_mat_op Bop,`
`const vsip_cscalar_f beta,`
`const vsip_cmview_f * R);`
	Calculate the general product of two matrices and accumulate.

`void`
`vsip_gems_f(`
`const vsip_scalar_f alpha,`
`const vsip_mview_f * A,`
`const vsip_mat_op Aop,`
`const vsip_scalar_f beta,`
`const vsip_mview_f * C);`
	Calculate a general matrix sum.

`void`
`vsip_cgems_f(`
`const vsip_cscalar_f alpha,`
`const vsip_cmview_f * A,`
`const vsip_mat_op Aop,`
`const vsip_cscalar_f beta,`
`const vsip_cmview_f * C);`
	Calculate a general matrix sum.

`void`
`vsip_Dmprod_P(`
`const vsip_Dmview_P * A,`
`const vsip_Dmview_P * B,`
`const vsip_Dmview_P * R);`
	Calculate the product of two matrices. The following instances are supported:
	`vsip_mprod_f`
	`vsip_mprod_i`
	`vsip_mprod_si`
	`vsip_cmprod_f`
	`vsip_cmprod_i`
	`vsip_cmprod_si`

Prototype	Description
void vsip_cmprodh_P(const vsip_cmview_P * A, const vsip_cmview_P * B, const vsip_cmview_P * R);	Calculate the product a complex matrix and the Hermitian of a complex matrix. The following instances are supported: vsip_cmprodh_f vsip_cmprodh_i vsip_cmprodh_si
void vsip_cmprodj_P(const vsip_cmview_P * A, const vsip_cmview_P * B, const vsip_cmview_P * R);	Calculate the product a complex matrix and the conjugate of a complex matrix. The following instances are supported: vsip_cmprodj_f vsip_cmprodj_i vsip_cmprodj_si
void vsip_Dmprodt_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);	Calculate the product of a matrix and the transpose of a matrix. The following instances are supported: vsip_mprodt_f vsip_mprodt_i vsip_mprodt_si vsip_cmprodt_f vsip_cmprodt_i vsip_cmprodt_si
void vsip_Dmvprod_P(const vsip_Dmview_P * A, const vsip_Dvview_P * X, const vsip_Dvview_P * Y);	Calculate a matrix–vector product. The following instances are supported: vsip_mvprod_f vsip_mvprod_i vsip_mvprod_si vsip_cmvprod_f vsip_cmvprod_i vsip_cmvprod_si
void vsip_Dmtrans_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);	Transpose a matrix. The following instances are supported: vsip_mtrans_bl vsip_mtrans_f vsip_mtrans_i vsip_mtrans_si vsip_cmtrans_f vsip_cmtrans_i vsip_cmtrans_si
vsip_Dscalar_P vsip_Dvdot_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B);	Compute the inner (dot) product of two vectors. The following instances are supported: vsip_vdot_f vsip_cvdot_f

Prototype	Description
void vsip_Dvmprod_P(const vsip_Dvview_P * X, const vsip_Dmview_P * A, const vsip_Dvview_P * Y);	Calculate a vector–matrix product. The following instances are supported: vsip_vmprod_f vsip_vmprod_i vsip_vmprod_si vsip_cvmprod_f vsip_cvmprod_i vsip_cvmprod_si
void vsip_vouter_f(const vsip_scalar_f alpha, const vsip_vview_f * X, const vsip_vview_f * Y, const vsip_vview_f * R);	Calculate the outer product of two vectors.
void vsip_cvouter_f(const vsip_cscalar_f alpha, const vsip_cvview_f * X, const vsip_cvview_f * Y, const vsip_cvview_f * R);	Calculate the outer product of two vectors.

6.2 Special Linear System Solvers

Prototype	Description
int vsip_covsol_f(const vsip_mview_f * A, const vsip_mview_f * XB);	Solve a covariance linear system problem.
int vsip_ccovsol_f(const vsip_cmview_f * A, const vsip_cmview_f * XB);	Solve a covariance linear system problem.
int vsip_llsqsol_f(const vsip_mview_f * A, const vsip_mview_f * XB);	Solve a linear least squares problem.
int vsip_cllsqsol_f(const vsip_cmview_f * A, const vsip_cmview_f * XB);	Solve a linear least squares problem.
int vsip_toepsol_f(const vsip_vview_f * T, const vsip_vview_f * B, const vsip_vview_f * W, const vsip_vview_f * X);	Solve a real symmetric positive definite Toeplitz linear system.
int vsip_ctoepsol_f(const vsip_cvview_f * T, const vsip_cvview_f * B, const vsip_cvview_f * W, const vsip_cvview_f * X);	Solve a real symmetric positive definite Toeplitz linear system.

6.3 General Square Linear System Solver

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

6.4 Symmetric Positive Definite Linear System Solver

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

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

6.5 Overdetermined Linear System Solver

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

6.6 Extensions

Prototype	Description
void vsip_Dminvlu_P(const vsip_Dmview_P * A, const vsip_vview_i * V, const vsip_Dmview_P * R);	Invert a square matrix using LU decomposition. The following instances are supported: vsip_minvlu_f vsip_cminvlu_f