GE
Intelligent Platforms

Software Reference Manual

Radstone Signal Processing Library (RSPL) Manual

8th Edition

$$length = N$$

$$\hat{y}[k] := \sum_{j=0}^{M-1} r[j]\langle x[k + j - \lfloor M/2 \rfloor]^* \rangle \quad for \quad 0 \le k \le N-1$$

$$y[k] := \hat{y}[k]^* \begin{cases} 1/(k + \lceil M/2 \rceil) : 0 \le k < \lfloor M/2 \rfloor \\ 1/M : \lfloor M/2 \rfloor \le k < N - \lfloor M/2 \rfloor \\ 1/(N + \lceil M/2 \rceil - 1 - k) : N - \lfloor M/2 \rfloor \le k < N \end{cases}$$

Minimum (not zero padded):

$$length = N - M + 1$$

$$\hat{y}[k] := \sum_{j=0}^{M-1} r[j]\langle x[k + j]^* \rangle \quad for \quad 0 \le k \le N - M$$

$$y[k] := \hat{y}[k] / M$$

$$\text{Where } \langle x[j] \rangle = \begin{cases} x[j] \; 0 \le j < N \\ 0 : otherwise \end{cases}$$

Contents

1 • Introduction

This section describes the functions available in the GE Intelligent Platforms library of vector, signal processing, and linear algebra routines.

1.1 Types

The following base types are available:

`rad_bool`	boolean
`rad_cscalar_f`	complex floating-point scalar
`rad_cscalar_i`	complex integer scalar
`rad_cscalar_si`	complex short integer scalar
`rad_index`	index to a vector
`rad_index_mi`	index (pair of integers) to a matrix

The library also passes information around in abstract data types. These structures are not opaque and the internal values are not hidden from the user, but they should only be created and destroyed with library functions that reference them via a pointer. There are no get attribute functions to access the internal values.

1.2 Symbols and Flags

The following symbolic constants are defined.

```
RAD_TRUE  RAD_FALSE
```

Other symbols are defined in enumerated types. The valid choices are listed with each function description.

1.3 Vector Variables

A vector is passed into a function with three parameters:

- a pointer to the start of the data
- an integer for the distance between processed elements (stride); stride = 1 (unit stride) means every element of a vector has to be processed, stride = 2 means every other element, and so on
- an integer for the number of elements accessed (length).

Stride and length are measured in number of data elements (not bytes). For a compound type like complex, this means the number of (real, imaginary) pairs and not the number of atomic type elements (floats for example).

When several vectors are passed into a function, each has its own stride, but usually they all share a common length. In the function prototypes, each vector variable has two parameters (a data pointer followed by a stride) and the common processed length is usually the last parameter.

When a stride is negative the processing progresses backwards through the data so a suitable offset must be added to the data pointer. With proper pointer arithmetic the offset is also measured in number of data elements.

1.4 Matrix Variables

Matrices are stored in row-major format. A matrix is passed into a function with four parameters:

- a pointer to the start of the data
- an integer for the distance between adjacent elements in a column of the matrix (leading dimension); it is denoted by ldM for matrix M; it can be viewed as the number of columns in the matrix containing M;
- an integer for the number or rows
- an integer for the number or columns (less than or equal to the leading dimension).

As with vectors, the leading dimension is measured in number of data elements.

When several matrices are passed into a function, each has its own leading dimension, but often they all have the same dimensions, or have one dimension in common. In the function prototypes, each matrix variable has two parameters (a data pointer followed by a leading dimension); the dimension parameters (numbers of row and columns) are often common for several matrices and are usually the last in the parameter list. The description of each function explains the dimensions of the matrices.

1.5 Complex Variables

Most functions that use complex data are available in two versions: one accepts data stored in interleaved format and each complex variable is passed as a single parameter; the other takes data stored in split format and each complex variable is passed via two parameters one each for the real and imaginary parts.

1.6 Function Names

For almost all functions, the suffix indicates the base datatype of the arguments as follows:

`i`	integer
`bl`	boolean
`si`	short integer
`vi`	vector index
`mi`	matrix index
`f`	float

1.7 Hints

The following mechanisms are provided for the programmer to indicate preferences for optimization: Currently they are all ignored but are reserved for future use. The only exception is the algorithm hint in FIR filters.

Flags of the enumerated type `rad_memory_hint` specified when allocating or creating some objects.

Flags of the enumerated type `rad_alg_hint` used to indicate whether algorithmic optimization should minimize execution time, memory use, or maximize numerical accuracy.

An indication of how many times an object will be used (filters and FFTs have such a parameter).

1.8 Notation

The following standard mathematical notation is used in the function descriptions.

$:=$	assignment operator
i	square root of -1
$\lvert x \rvert$	absolute value of the real number x
$\lvert z \rvert$	modulus of the complex number z
$\lfloor x \rfloor$	floor of the real number x (largest integer less than or equal to x)
$\lceil x \rceil$	ceiling of the real number x (smallest integer greater than or equal to x)
z^*	conjugate of the complex number z
M^{T}	transpose of the matrix M
M^{H}	Hermitian (conjugate transpose) of the complex matrix M

Note that in expressions i is always the square root of -1; vectors and matrices are indexed with j and k.

An elementwise operation on vectors will be written $C[j * \texttt{strideC}] := A[j * \texttt{strideA}] + B[j * \texttt{strideB}]$. Often the range of the index variable is not given explicity; in such cases it is clear from the context that it runs over all the elements in the vectors and that the lengths of the vectors must be equal.

An M by N matrix has M rows and N columns.

1.9 Errors and Restrictions

Many functions require that their arguments be conformant. This means that the objects passed have compatible attributes: for example, size and shape of matrices, lengths of vectors or filter kernels.

If an argument is required to be valid, it means:

- a pointer is not `NULL`
- a flag is a member of the required enumerated type
- an object has been initialized and not destroyed.

Errors can occur for the following reasons:

1. an argument is outside the domain for calculation
2. over/underflow during calculation
3. failure to allocate memory
4. algorithm failure because of inappropriate data (as when a matrix does not have full rank)
5. arguments are invalid, out of range, or non-conformant.

Only errors of type 5 are regarded as fatal: in this case, the development version of the library will write a message to stderr and call exit.

Errors of types 3 and 4 are signaled through the return value of the function. A create function will return `NULL` if the allocation fails; functions with integer return codes use zero to indicate success. The calling program is not alerted to errors of types 1 and 2.

2 • Getting the Best Performance

This section is a short guide for programmers using the RSPL Library. It contains explanations of library behavior, and tips on selecting the right storage options for your data to increase performance.

2.1 Version Information

Information about the version of the RSPL library you are using can be found in the comments at the top of the include file `vsip.h`. There is no way that a program can determine the library version at run-time.

2.2 Memory Alignment

The efficiency of many operations is improved if data within memory is correctly aligned on certain word boundaries.

Vectors and matrices can be loaded and stored faster if they are vector aligned.

The following table gives the vector alignment and minimum vector length for float data:

Technology	Vector Aligment
SSE	16
AVX	32
AltiVec	16

Alignment can be controlled using a function such as `memalign`. This is a C function that is not in the ANSI standard but is available on many systems. It is defined in `malloc.h` on Linux systems.

The following macro redefines `malloc` so that all memory allocation is optimally aligned:

```
#include <malloc.h>
#define malloc(SIZE) memalign(16, SIZE)
```

Some operating systems (*e.g.* Apple's OSX) automatically align all memory to a 16-byte boundary so `memalign` is not needed.

2.3 Vector/Matrix Format

When available, vector and matrix calculations are done using single instruction, multiple data (SIMD) instructions to process several elements simultaneously. This imposes a minimum vector length given in the table below:

Technology Vector Length (floats)	minimum
SSE	4
AVX	8
AltiVec	4

For short ints (16 bits) the vector length should be twice that of the float vector length. If the vector unit supports doubles (64 bits), then the vector length should be half that of floats.

For best performance all input and output vectors should:

- have a stride of 1

  Vectors and matrices can be loaded and stored much quicker when they are contiguous in memory. The library includes special optimisations for a stride length of 2 (which was added for interleaved complex numbers), but all other non-unit strides will be significantly slower than a stride of 1 and, in many cases, almost as slow as unvectorised scalar code. Note that, a stride of -1 will also be significantly slower than a stride of $+1$.

- be vector aligned

- have length greater than or equal to the vector length

  The vector unit works on arrays of the vector length so no speed up is gained by using the library on vectors of length less than this.

- have row (row major matrices) or column (column major matrices) length divisible by the vector length

  For a row major matrix: if the row length of a matrix is not divisible by the vector length then the alignment of the first element of each row will change for each row/column. For optimal performance the first element of each row should be vector aligned.

  The same rule applies to columns in column major matrices.

- have a length divisible by the vector length

  Any elements at the end of the vector which cannot be dealt with by the vector unit must be dealt with in normal scalar code, which will decrease the performance. The decrease in performance becomes less important for longer vectors.

2.4 Error Checking and Debugging

Two versions of the RSPL library are provided: a performance version and a development version. The development version of the library (signified by a 'D' in the library's name) contains full error checking and should always be used when developing and debugging applications.

A few library functions return status information: always check the return code of those that do.

The performance version of the library contains no error checking, and consequently runs faster than the development library. The performance library should only be used with applications that have been run successfully with the development version of the library.

When timing code, the performance version of the library should be used.

Note: the performance version of the library reads in data before it knows how much will be used and as a result often reads more data than is needed. This is not a problem, except when using memory checkers such as Electric Fence which object to this behavior. The development library only reads in the data it intends to use and so is safe to use with memory checkers.

2.5 Scalar Functions

As the vector unit works on arrays of the vector length, scalar functions in the library are not vectorized.

2.6 Random Number Generation

The random number generation functions have not been vectorized in the current version of the library.

2.7 Vector and Elementwise Operations

All vector and elementwise operations work optimally on vectors which match the conditions given in Section 2.3.

2.8 Signal Processing Functions

All signal processing operations work optimally on vectors which match the conditions given in Section 2.3.

Most of the signal processing routines are split into three stages:

- a create stage
- a compute stage
- a destroy stage.

The library has been optimized to minimize the time taken to do the compute stage, which means as much precomputation as possible is done in the create stage. If you are using the same signal processing routine on many vectors of the same length, it is far quicker to just create the required signal processing object once and reuse it for each computation stage rather than recreating the object each time it is needed.

2.9 FFT Functions

To get the best performance from an FFT, a vector must have length a multiple of the numbers 2, 4, 8, and 3 only. If a vector length is not a multiple of these numbers, a DFT may be done, which is considerably slower than an FFT. Factors of 3 should be avoided if possible. An FFT will only be done for factors of 3 if the length also has a factor of 16, otherwise a DFT is done.

When doing large FFT's, optimal routines have been developed for the lengths: 4096, 8192, 16384, 32768, and 65536. These lengths should be much quicker than lengths of similar magni-

tude. In-place FFT's are normally faster than out-of-place FFT's. FFT's are fastest with a scale factor of 1. However, if you need to use a different scale factor, it is better to let the FFT routine do the scaling rather than to do it yourself.

The internal FFT routines only work on vector aligned data with a stride of 1. If vectors are used which do not match these restrictions an internal copy of the vector will be made. This is an important consideration when using large vectors. Also, if complex vectors are not stored split an internal copy will be made.

The current version of the RSPL library does not have any special FFT routines for doing multiple FFT's, so the time to do n single FFT's will be approximately the same as using the multiple FFT routines on a matrix of n rows.

The `ntimes` parameter to the FFT functions is ignored. The algorithmic hint is only used in the FFT create function: if the `RAD_ALG_NOISE` hint is used, the FFT create function will take significantly longer. By default, the algorithms are optimized to minimize execution time.

2.10 FIR Filter, Convolution and Correlation Functions

These functions call the FFT functions internally and are therefore subject to the same restrictions.

Hints are ignored with the exception of the internal calling of the FFT create function described in the FFT functions section.

2.11 Linear Algebra Functions

For optimal performance the vectors and matrices used with the linear algebra functions should match the conditions given in Section 2.3. (See also the sections below when using complex LU, complex Cholesky, or complex QRD functions).

2.12 Matrix and Vector Operations

Matrix and vector operations should work optimally on row or column major matrices (row major is the default), however, the restriction exists that all matrices passed to a function should be of the same order. For example, using two row major matrices as input to a function and a column major as output will be slower than using all row major or all column major. When matrices are passed to RSPL functions that are not all of the same order, the library will assume they are all row major and treat the column major matrices as strided matrices. (See also the sections below when using LU, Cholesky, or QRD functions).

2.13 LU Decomposition, Cholesky and QRD Functions

These functions have three separate stages:

- a create stage
- a compute stage
- a destroy stage.

The library has been optimized to minimize the time taken to do the compute stage, which means as much precomputation as possible is done in the create stage. If you are using the linear algebra routine on many matrices of the same size, it is far quicker to just create the required linear algebra object once and reuse it for each computation stage rather than recreating the object each time it is needed.

If matrices of different orders or strided matrices are passed to these functions, an internal copy will be made of the entire matrix before the computation is done. This is an important consideration when using large matrices. (Note: unaligned matrices do NOT require internal copying provided they have a stride of one and all matrices used with the functions are of the same order).

When using the QRD functions, it is only necessary to save the Q matrix if using the `qrdprodq` function; the `qrsol` and `qrdsolr` do not need the Q matrix.

2.14 Special Linear System Solvers

The `covsol` and `llsqsol` functions internally use the QRD functions and so have the same requirements for optimal performance.

The `toepsol` functions are based on vector operations and so have the same requirements for optimal performance.

2.15 Controlling the Number of Threads

The RSPL library is multithreaded and will take advantage of multiple cores on the processor invoking it. Utilizing multiple threads is automatic:

- The maximum number of threads used is set when `rad_init` is called.
- The maximum number running at any one time is also set at that point. If a threaded routine is called with (say) 4 threads and we have hit this maximum number running then four of them are shut down before the new function is executed.
- The number of threads invoked when a routine is called, is decided by that routine by reference to the data (vector or matrix) size specified in the call, to provide the best performance for that call.

It is possible to change the maximum number of threads used.

1. A threaded, and a non-threaded ("serial") version of the library are provided. If you wish to only ever use one thread in a library call, use the serial version of the library.
2. The maximum number of threads used for a specific function call, and the maximum number kept running at any one time, can be changed by a call to the routine `Thread_SetParams` with arguments `num_threads` and `max_num_running`. This call, if used, *must* be made before the library initialization routine `rad_init` is called.
3. When calling the routine `Thread_SetParams` the value of `max_num_running` must be greater or equal to 3*`num_threads`. If the user enters a smaller value than this in their `Thread_SetParams` function call then the function will set the value of `max_num_running` to 3*`num_threads`.

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

3 • Support Functions

3.1 Initialization and Finalization

rad_init

rad_finalize

rad_init

Provides initialization, allowing the library to allocate and set a global state, and prepare to support the use of RSPL functionality by the user.

Prototype

```
int rad_init(
                    void *ptr);
```

Parameters

`ptr`, pointer to structure, input.

Return Value

Error code.

Description

This required function informs the RSPL library that library initialization is requested, and that other RSPL functions will be called. This function must be called at least once by all RSPL programs. It may be called multiple times as well, with corresponding calls to `rad_finalize` to create nested pairs of initialization/termination. Only the `rad_finalize` matching the first `rad_init` call will actually release the library. Intermediate calls to `rad_init` have no effect, but support easy program/library development through compositional programming, where the user may not even know that a library itself invokes RSPL.
The argument is reserved for future purposes. The `NULL` pointer should be passed for RSPL 1.0 compliance.
Returns zero if the initialization succeeded, and non-zero otherwise.

Restrictions

This function may be called at any time during the execution of the program.

Errors

Notes

All programs must use the initialization function before calling any other RSPL functions.
Unsuccessful initialization of the library is not an error. It is always signalled via the function's return value, and should always be checked by the application.

rad_finalize

Provides cleanup and releases resources used by RSPL (if the last of a nested series of calls), allowing the library to guarantee that any resources allocated by rad_init are no longer in use after the call is complete.

Prototype

```
int rad_finalize(
                    void *ptr);
```

Parameters

ptr, pointer to structure, input.

Return Value

Error code.

Description

This required function informs the RSPL library that it is no longer being used by a program, so that all needed global state and hardware state can be returned. All programs must call this function at least once if they terminate. If the program does terminate, the last RSPL function called must be an outermost rad_finalize. Because nested rad_init's are supported, so are nested rad_finalize's.
The user must explicitly destroy all RSPL objects before calling this function if this is an 'outermost' rad_finalize. When nesting initializations, there is no need to destroy all objects prior to calling this function, but the user is obliged to keep track of the nesting depth if programs are written in such a manner.
Returns zero if the finalization succeeded, and non-zero otherwise. Zero is always returned if the call is not outermost.

Restrictions

This function may only be called if a previous rad_init call has been made, with no previous corresponding rad_finalize.

Errors

An outermost rad_finalize call produces an error if there are any RSPL objects not destroyed.

Notes

The user program is always responsible for returning resources it is no longer using by destroying RSPL objects. An outermost finalization function will return resources that it allocated previously with rad_init. Non-outermost rad_finalize's always return zero (success).

3.2 Sundry Functions

rad_complete

rad_cstorage

rad_complete

Force all deferred RSPL execution to complete.

Prototype

```
void rad_complete(void);
```

Parameters

none.

Return Value

none.

Description

Forces all deferred RSPL execution to complete and then returns. NOTE: there is no deferred execution in this implementation of RSPL.

Restrictions

Errors

Notes

rad_cstorage

Returns the preferred complex storage format for the system.

Prototype

```
rad_cmplx_mem rad_cstorage(void);
```

Parameters

none.

Return Value

enumerated type.

RAD_CMPLX_INTERLEAVED	interleaved
RAD_CMPLX_SPLIT	split
RAD_CMPLX_NONE	no preference

Description

Returns the preferred complex storage format for the system.

Restrictions

Errors

Notes

4 • Scalar Functions

4.1 Real Scalar Functions

rad_acos_f

Computes the principal radian value in $[0, \pi]$ of the inverse cosine of a scalar.

Prototype

```
float  rad_acos_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\cos^{-1}(A)$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

rad_asin_f

Computes the principal radian value in $[0, \pi]$ of the inverse sine of a scalar.

Prototype

```
float  rad_asin_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\sin^{-1}(A)$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

rad_atan_f

Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent of a scalar.

Prototype

```
float  rad_atan_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\tan^{-1}(A)$.

Restrictions

Errors

Notes

rad_atan2_f

Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of two scalars.

Prototype

```
float  rad_atan2_f(
                   float  A,
                   float  B);
```

Parameters

A, real scalar, input.

B, real scalar, input.

Return Value

real scalar.

Description

return value := $\tan^{-1}(A/B)$.
The rules for calculating the function value are the same as those for the ANSI C function `atan2`.

Restrictions

The arguments must not be both zero.

Errors

Notes

rad_ceil_f

Computes the ceiling of a scalar.

Prototype

```
float  rad_ceil_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\lceil A \rceil$.
Returns the smallest integer greater than or equal to the argument.

Restrictions

Errors

Notes

rad_cos_f

Computes the cosine of a scalar angle in radians.

Prototype

```
float  rad_cos_f(
                    float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := cos(A).

Restrictions

Errors

Notes

Input argument is expressed in radians.

rad_cosh_f

Computes the hyperbolic cosine of a scalar.

Prototype

```
float  rad_cosh_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := cosh(A).

Restrictions

Errors

Notes

rad_exp_f

Computes the exponential of a scalar.

Prototype

```
float  rad_exp_f(
                    float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := exp(A).

Restrictions

Overflow will occur if the argument is greater than the natural logarithm of the maximum representable number. Underflow will occur if the argument is less than the natural logarithm of the maximum representable number.

Errors

Notes

rad_floor_f

Computes the floor of a scalar.

Prototype

```
float  rad_floor_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\lfloor A \rfloor$.
Returns the largest integer less than or equal to the argument.

Restrictions

Errors

Notes

rad_log_f

Computes the natural logarithm of a scalar.

Prototype

```
float  rad_log_f(
                    float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\log_e(A)$.

Restrictions

The agument must be greater than zero.

Errors

Notes

rad_log10_f

Computes the base 10 logarithm of a scalar.

Prototype

```
float  rad_log10_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\log_{10}(A)$.

Restrictions

The argument must be greater than zero.

Errors

Notes

rad_mag_f

Computes the magnitude (absolute value) of a scalar.

Prototype

```
float  rad_mag_f(
                    float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := |A|.

Restrictions

Errors

Notes

rad_pow_f

Computes the power function of two scalars.

Prototype

```
float  rad_pow_f(
                   float  A,
                   float  B);
```

Parameters

A, real scalar, input.

B, real scalar, input.

Return Value

real scalar.

Description

return value := A^B.

Restrictions

Errors

Notes

rad_sin_f

Computes the sine of a scalar angle in radians.

Prototype

```
float  rad_sin_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := sin(A).

Restrictions

Errors

Notes

Input argument is expressed in radians.

rad_sinh_f

Computes the hyperbolic sine of a scalar.

Prototype

```
float  rad_sinh_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := sinh(A).

Restrictions

Errors

Notes

rad_sqrt_f

Computes the square root of a scalar.

Prototype

```
float  rad_sqrt_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := $\sqrt{A}$.

Restrictions

The argument must be greater than or equal to zero.

Errors

Notes

rad_tan_f

Computes the tangent of a scalar angle in radians.

Prototype

```
float  rad_tan_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := tan(A).

Restrictions

For values $(n + 1/2)\pi$, the tangent function has a singularity and is undefined.

Errors

Notes

Input argument is expressed in radians.

rad_tanh_f

Computes the hyperbolic tangent of a scalar.

Prototype

```
float  rad_tanh_f(
                   float  A);
```

Parameters

A, real scalar, input.

Return Value

real scalar.

Description

return value := tanh(A).

Restrictions

Errors

Notes

4.2 Complex Scalar Functions

rad_arg_f

rad_CADD_f

rad_cadd_f

rad_RCADD_f

rad_rcadd_f

rad_CDIV_f

rad_cdiv_f

rad_CRDIV_f

rad_crdiv_f

rad_CEXP_f

rad_cexp_f

rad_CJMUL_f

rad_cjmul_f

rad_cmag_f

rad_cmagsq_f

rad_CMPLX_f

rad_cmplx_f

rad_CMUL_f

rad_cmul_f

rad_RCMUL_f

rad_rcmul_f

rad_CNEG_f

rad_cneg_f

rad_CONJ_f

rad_conj_f

rad_CRECIP_f

rad_crecip_f

rad_CSQRT_f

rad_csqrt_f

rad_CSUB_f

rad_csub_f

rad_RCSUB_f

rad_rcsub_f

rad_CRSUB_f

rad_crsub_f

rad_imag_f

rad_polar_f

rad_real_f

rad_RECT_f

rad_rect_f

rad_arg_f

Returns the argument in radians $[-\pi, \pi]$ of a complex scalar.

Prototype

```
float  rad_arg_f(
                    void  *x);
```

Parameters

x, complex scalar, input.

Return Value

real scalar.

Description

return value := $\tan^{-1}(\text{imag}(x)/\text{real}(x))$.

Restrictions

The argument must not be zero.

Errors

Notes

rad_CADD_f

Computes the complex sum of two scalars.

Prototype

```
void rad_CADD_f(
                    void  *x,
                    void  *y,
                    void  *z);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x + y$.

Restrictions

Errors

Notes

rad_cadd_f

Computes the complex sum of two scalars.

Prototype

```
rad_cscalar_f rad_cadd_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := x + y.

Restrictions

Errors

Notes

rad_RCADD_f

Computes the complex sum of two scalars.

Prototype

```
void rad_RCADD_f(
                    float   x,
                    void   *y,
                    void   *z);
```

Parameters

x, real scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x + y$.

Restrictions

Errors

Notes

rad_rcadd_f

Computes the complex sum of two scalars.

Prototype

```
rad_cscalar_f rad_rcadd_f(
                    float   x,
                    void   *y);
```

Parameters

x, real scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := x + y.

Restrictions

Errors

Notes

rad_CDIV_f

Computes the complex quotient of two scalars.

Prototype

```
void rad_CDIV_f(
                    void  *x,
                    void  *y,
                    void  *z);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x/y$.

Restrictions

The divisor must not be zero.

Errors

Notes

rad_cdiv_f

Computes the complex quotient of two scalars.

Prototype

```
rad_cscalar_f rad_cdiv_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := x/y.

Restrictions

The divisor must not be zero.

Errors

Notes

rad_CRDIV_f

Computes the complex quotient of two scalars.

Prototype

```
void rad_CRDIV_f(
                    void   *x,
                    float   y,
                    void   *z);
```

Parameters

x, complex scalar, input.

y, real scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x/y$.

Restrictions

The divisor must not be zero.

Errors

Notes

rad_crdiv_f

Computes the complex quotient of two scalars.

Prototype

```
rad_cscalar_f rad_crdiv_f(
                    void   *x,
                    float   y);
```

Parameters

x, complex scalar, input.

y, real scalar, input.

Return Value

complex scalar.

Description

return value := x/y.

Restrictions

The divisor must not be zero.

Errors

Notes

rad_CEXP_f

Computes the exponential of a scalar.

Prototype

```
void rad_CEXP_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, pointer to complex scalar, output.

Return Value

none.

Description

$*y := \exp(x)$.

Restrictions

Overflow will occur if the argument is greater than the natural logarithm of the maximum representable number. Underflow will occur if the argument is less than the natural logarithm of the maximum representable number.

Errors

Notes

rad_cexp_f

Computes the exponential of a scalar.

Prototype

```
rad_cscalar_f rad_cexp_f(
                    void  *A);
```

Parameters

A, complex scalar, input.

Return Value

complex scalar.

Description

return value := exp(A).

Restrictions

Overflow will occur if the argument is greater than the natural logarithm of the maximum representable number. Underflow will occur if the argument is less than the natural logarithm of the maximum representable number.

Errors

Notes

rad_CJMUL_f

Computes the product a complex scalar with the conjugate of a second complex scalar.

Prototype

```
void rad_CJMUL_f(
                    void  *x,
                    void  *y,
                    void  *z);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x \cdot y^*$.

Restrictions

Errors

Notes

rad_cjmul_f

Computes the product a complex scalar with the conjugate of a second complex scalar.

Prototype

```
rad_cscalar_f rad_cjmul_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := $x \cdot y^*$.

Restrictions

Errors

Notes

rad_cmag_f

Computes the magnitude (absolute value) of a scalar.

Prototype

```
rad_cscalar_f rad_cmag_f(
                    void  *A);
```

Parameters

A, complex scalar, input.

Return Value

complex scalar.

Description

return value := |A|.

Restrictions

Errors

Notes

rad_cmagsq_f

Computes the magnitude squared of a complex scalar.

Prototype

```
float  rad_cmagsq_f(
                   void  *x);
```

Parameters

x, complex scalar, input.

Return Value

real scalar.

Description

return value := $|x|^2$.

Restrictions

Errors

Notes

rad_CMPLX_f

Form a complex scalar from two real scalars.

Prototype

```
void rad_CMPLX_f(
                    float   a,
                    float   b,
                    void   *r);
```

Parameters

a, real scalar, input.

b, real scalar, input.

r, pointer to complex scalar, output.

Return Value

none.

Description

$*r := a + i \cdot b$.

Restrictions

Errors

Notes

rad_cmplx_f

Form a complex scalar from two real scalars.

Prototype

```
rad_cscalar_f rad_cmplx_f(
                    float  re,
                    float  im);
```

Parameters

re, real scalar, input.

im, real scalar, input.

Return Value

complex scalar.

Description

return value := re + i · im.

Restrictions

Errors

Notes

rad_CMUL_f

Computes the complex product of two scalars.

Prototype

```
void rad_CMUL_f(
                    void  *x,
                    void  *y,
                    void  *z);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x \cdot y$.

Restrictions

Errors

Notes

rad_cmul_f

Computes the complex product of two scalars.

Prototype

```
rad_cscalar_f rad_cmul_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := $x \cdot y$.

Restrictions

Errors

Notes

rad_RCMUL_f

Computes the complex product of two scalars.

Prototype

```
void rad_RCMUL_f(
                    float   x,
                    void   *y,
                    void   *z);
```

Parameters

x, real scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x \cdot y$.

Restrictions

Errors

Notes

rad_rcmul_f

Computes the complex product of two scalars.

Prototype

```
rad_cscalar_f rad_rcmul_f(
                    float   x,
                    void   *y);
```

Parameters

x, real scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := $x \cdot y$.

Restrictions

Errors

Notes

rad_CNEG_f

Computes the negation of a complex scalar.

Prototype

```
void rad_CNEG_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, pointer to complex scalar, output.

Return Value

none.

Description

$*y := -x$.

Restrictions

Errors

Notes

rad_cneg_f

Computes the negation of a complex scalar.

Prototype

```
rad_cscalar_f rad_cneg_f(
                    void  *x);
```

Parameters

x, complex scalar, input.

Return Value

complex scalar.

Description

return value := $-x$.

Restrictions

Errors

Notes

rad_CONJ_f

Computes the complex conjugate of a scalar.

Prototype

```
void rad_CONJ_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, pointer to complex scalar, output.

Return Value

none.

Description

$*y := x^*$.

Restrictions

Errors

Notes

rad_conj_f

Computes the complex conjugate of a scalar.

Prototype

```
rad_cscalar_f rad_conj_f(
                    void  *x);
```

Parameters

x, complex scalar, input.

Return Value

complex scalar.

Description

return value := x^*.

Restrictions

Errors

Notes

rad_CRECIP_f

Computes the reciprocal of a complex scalar.

Prototype

```
void rad_CRECIP_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, pointer to complex scalar, output.

Return Value

none.

Description

$*y := 1/x$.

Restrictions

The argument must not be zero.

Errors

Notes

rad_crecip_f

Computes the reciprocal of a complex scalar.

Prototype

```
rad_cscalar_f rad_crecip_f(
                    void  *x);
```

Parameters

x, complex scalar, input.

Return Value

complex scalar.

Description

return value := $1/x$.

Restrictions

The argument must not be zero.

Errors

Notes

rad_CSQRT_f

Computes the square root a complex scalar.

Prototype

```
void rad_CSQRT_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, pointer to complex scalar, output.

Return Value

none.

Description

$*y := \sqrt{x}$.

Restrictions

Errors

Notes

rad_csqrt_f

Computes the square root a complex scalar.

Prototype

```
rad_cscalar_f rad_csqrt_f(
                    void  *x);
```

Parameters

x, complex scalar, input.

Return Value

complex scalar.

Description

return value := $\sqrt{x}$.

Restrictions

Errors

Notes

rad_CSUB_f

Computes the complex difference of two scalars.

Prototype

```
void rad_CSUB_f(
                    void  *x,
                    void  *y,
                    void  *z);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x - y$.

Restrictions

Errors

Notes

rad_csub_f

Computes the complex difference of two scalars.

Prototype

```
rad_cscalar_f rad_csub_f(
                    void  *x,
                    void  *y);
```

Parameters

x, complex scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := x − y.

Restrictions

Errors

Notes

rad_RCSUB_f

Computes the complex difference of two scalars.

Prototype

```
void rad_RCSUB_f(
                    float   x,
                    void   *y,
                    void   *z);
```

Parameters

x, real scalar, input.

y, complex scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x - y$.

Restrictions

Errors

Notes

rad_rcsub_f

Computes the complex difference of two scalars.

Prototype

```
rad_cscalar_f rad_rcsub_f(
                    float   x,
                    void   *y);
```

Parameters

x, real scalar, input.

y, complex scalar, input.

Return Value

complex scalar.

Description

return value := x − y.

Restrictions

Errors

Notes

rad_CRSUB_f

Computes the complex difference of two scalars.

Prototype

```
void rad_CRSUB_f(
                    void   *x,
                    float   y,
                    void   *z);
```

Parameters

x, complex scalar, input.

y, real scalar, input.

z, pointer to complex scalar, output.

Return Value

none.

Description

$*z := x - y$.

Restrictions

Errors

Notes

rad_crsub_f

Computes the complex difference of two scalars.

Prototype

```
rad_cscalar_f rad_crsub_f(
                    void    *x,
                    float    y);
```

Parameters

x, complex scalar, input.

y, real scalar, input.

Return Value

complex scalar.

Description

return value := x − y.

Restrictions

Errors

Notes

rad_imag_f

Extract the imaginary part of a complex scalar.

Prototype

```
float  rad_imag_f(
                   void  *x);
```

Parameters

x, complex scalar, input.

Return Value

real scalar.

Description

return value := imag(x).

Restrictions

Errors

Notes

rad_polar_f

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.

Prototype

```
void rad_polar_f(
                    void   *a,
                    float  *r,
                    float  *t);
```

Parameters

a, complex scalar, input.

r, pointer to real scalar, output.

t, pointer to real scalar, output.

Return Value

none.

Description

r := |a| and t := arg(a).

Restrictions

The argument must be non-zero.

Errors

Notes

Complex numbers are always stored in rectangular $x + iy$ format. The polar form is represented by two real scalars.

rad_real_f

Extract the real part of a complex scalar.

Prototype

```
float  rad_real_f(
                   void  *x);
```

Parameters

x, complex scalar, input.

Return Value

real scalar.

Description

return value := real(x).

Restrictions

Errors

Notes

rad_RECT_f

Convert a pair of real scalars from complex polar to complex rectangular form.

Prototype

```
void rad_RECT_f(
                    float   radius,
                    float   theta,
                    void   *r);
```

Parameters

`radius`, real scalar, input.

`theta`, real scalar, input.

`r`, pointer to complex scalar, output.

Return Value

none.

Description

$*\texttt{r} := \texttt{radius} \cdot (\cos(\texttt{theta}) + \mathrm{i} \cdot \sin(\texttt{theta}))$.

Restrictions

Errors

Notes

Complex numbers are always stored in rectangular $x + \mathrm{i}y$ format. The polar form is represented by two real scalars.

rad_rect_f

Convert a pair of real scalars from complex polar to complex rectangular form.

Prototype

```
rad_cscalar_f rad_rect_f(
                    float  r,
                    float  t);
```

Parameters

r, real scalar, input.

t, real scalar, input.

Return Value

complex scalar.

Description

return value := $\mathtt{r} \cdot (\cos(\mathtt{t}) + \mathrm{i} \cdot \sin(\mathtt{t}))$.

Restrictions

Errors

Notes

Complex numbers are always stored in rectangular $x + \mathrm{i}y$ format. The polar form is represented by two real scalars.

4.3 Index Scalar Functions

rad_MATINDEX

rad_matindex

rad_mcolindex

rad_mrowindex

rad_MATINDEX

Form a matrix index from two vector indices.

Prototype

```
void rad_MATINDEX(
                    unsigned int    r,
                    unsigned int    c,
                    rad_scalar_mi  *mi);
```

Parameters

r, vector-index scalar, input.

c, vector-index scalar, input.

mi, pointer to matrix-index scalar, output.

Return Value

none.

Description

$*mi := (r, c)$.

Restrictions

Errors

Notes

rad_matindex

Form a matrix index from two vector indices.

Prototype

```
rad_scalar_mi rad_matindex(
                    unsigned int  r,
                    unsigned int  c);
```

Parameters

r, vector-index scalar, input.

c, vector-index scalar, input.

Return Value

matrix-index scalar.

Description

return value := (r, c).

Restrictions

Errors

Notes

rad_mcolindex

Returns the column vector index from a matrix index.

Prototype

```
unsigned int  rad_mcolindex(
                    rad_scalar_mi mi);
```

Parameters

`mi`, matrix-index scalar, input.

Return Value

vector-index scalar.

Description

return value := column(`mi`).

Restrictions

Errors

Notes

rad_mrowindex

Returns the row vector index from a matrix index.

Prototype

```
unsigned int  rad_mrowindex(
                    rad_scalar_mi mi);
```

Parameters

`mi`, matrix-index scalar, input.

Return Value

vector-index scalar.

Description

return value := row(`mi`).

Restrictions

Errors

Notes

5 • Random Number Generation

5.1 Random Number Functions

rad_randcreate

Create a random number generator state object.

Prototype

```
rad_randstate * rad_randcreate(
                  unsigned int  seed,
                  unsigned int  numprocs,
                  unsigned int  id,
                  rad_rng       portable);
```

Parameters

`seed`, vector-index scalar, input. Seed to initialise the generator.

`numprocs`, vector-index scalar, input.

`id`, vector-index scalar, input.

`portable`, enumerated type, input.

`RAD_PRNG`	portable generator
`RAD_NPRNG`	non-portable generator

Return Value

structure.

Description

Creates a state object for use by a random number generation function. The random number generator is characterised by specifying the number of random number generators (`numprocs`) the application is expected to create, and the index (`id`) of this generator. If the portable sequence is specified, then the number of random number generators specifies how many subsequences the primary sequence is partitioned into.
The function returns a random state object which holds the state information for the random number sequence generator, or `NULL` if the create fails.

Restrictions
Errors

The arguments must conform to the following:

1. $0 < \texttt{id} \leqslant \texttt{numprocs} \leqslant 2^{31} - 1$.

Notes

You must call this function for each random number sequence/stream the application needs. This might be one per processor, one per thread, etc. For the portable sequence to have the desired pseudo-random properties, each create must specify the same number of processors/subsequences.

rad_randdestroy

Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure.

Prototype

```
int rad_randdestroy(
                    rad_randstate *rand);
```

Parameters

`rand`, structure, input.

Return Value

Error code.

Description

Destroys a random number state object.

Restrictions
Errors

The arguments must conform to the following:

1. The random number state object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_randu_f

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 $(0, 2^{31} - 1)$.

Prototype

```
float  rad_randu_f(
                    rad_randstate *state);
```

Parameters

`state`, structure, input.

Return Value

real scalar.

Description

return value := uniform$(0, 1)$.

Restrictions

Errors

Notes

rad_vrandu_f

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 $(0, 2^{31} - 1)$.

Prototype

```
void rad_vrandu_f(
                    rad_randstate *state,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

`state`, structure, input.

`R`, real vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] :=$ uniform$(0, 1)$.

Restrictions

Errors

Notes

rad_cvrandu_f

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 $(0, 2^{31} - 1)$.

Prototype

```
void rad_cvrandu_f (
                    rad_randstate *state,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

`state`, structure, input.

`R`, complex vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] :=$ uniform$(0, 1) +$ i $\cdot$ uniform$(0, 1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components where each component is uniform$(0, 1)$.

rad_cvrandu_split_f

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 $(0, 2^{31} - 1)$.

Prototype

```
void rad_cvrandu_split_f(
                    rad_randstate *state,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

`state`, structure, input.

`R_re`, real part of complex vector, length n, output.

`R_im`, imaginary part of complex vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] :=$ uniform$(0, 1) +$ i $\cdot$ uniform$(0, 1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components where each component is uniform$(0, 1)$.

rad_randn_f

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

```
float  rad_randn_f(
                    rad_randstate *state);
```

Parameters

`state`, structure, input.

Return Value

real scalar.

Description

return value := $\mathrm{N}(0, 1)$.

Restrictions

Errors

Notes

If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

rad_crandn_f

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

```
rad_cscalar_f rad_crandn_f(
                    rad_randstate *state);
```

Parameters

`state`, structure, input.

Return Value

complex scalar.

Description

return value := $\mathrm{N}(0,1) + \mathrm{i} \cdot \mathrm{N}(0,1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components that are uncorrelated.
If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

rad_vrandn_f

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

```
void rad_vrandn_f(
                  rad_randstate *state,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

`state`, structure, input.

`R`, real vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] := \mathrm{N}(0,1)$.

Restrictions

Errors

Notes

If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

rad_cvrandn_f

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

```
void rad_cvrandn_f(
                   rad_randstate *state,
                   void          *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

`state`, structure, input.

`R`, complex vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] := \mathrm{N}(0,1) + \mathrm{i} \cdot \mathrm{N}(0,1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components that are uncorrelated.
If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

rad_cvrandn_split_f

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

```
void rad_cvrandn_split_f(
                    rad_randstate *state,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

`state`, structure, input.

`R_re`, real part of complex vector, length n, output.

`R_im`, imaginary part of complex vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] := \mathrm{N}(0,1) + \mathrm{i} \cdot \mathrm{N}(0,1)$.

Restrictions

Errors

Notes

The complex random number has real and imaginary components that are uncorrelated.
If a true Gaussian random deviate is needed, the Box-Muller algorithm should be used. See Donald E. Knuth, Seminumerical Algorithms, 2nd ed., vol. 2, p117 of The Art of Computer Programming, Addison-Wesley, 1981.

6 • Vector And Elementwise Operations

6.1 Elementary Mathematical Functions

rad_vacos_f

rad_macos_f

rad_vasin_f

rad_masin_f

rad_vatan_f

rad_matan_f

rad_vatan2_f

rad_matan2_f

rad_vcos_f

rad_mcos_f

rad_vcosh_f

rad_mcosh_f

rad_vceil_f

rad_vexp_f

rad_cvexp_f

rad_cvexp_split_f

rad_mexp_f

rad_cmexp_f

rad_cmexp_split_f

rad_vexp10_f

rad_mexp10_f

rad_vfloor_f

rad_vlog_f

rad_cvlog_f

rad_cvlog_split_f

rad_mlog_f

rad_cmlog_f

rad_cmlog_split_f

rad_vlog10_f

rad_mlog10_f

rad_vsin_f

rad_msin_f

rad_vsinh_f

rad_msinh_f

rad_Dvsqrt_P

rad_cvsqrt_split_f

rad_Dmsqrt_P

rad_cmsqrt_split_f

rad_vtan_f

rad_mtan_f

rad_vtanh_f

rad_mtanh_f

rad_vacos_f

Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a vector.

Prototype

```
void rad_vacos_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] := \cos^{-1}($A$[j *$ strideA$])$ where $0 \leqslant j < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

rad_macos_f

Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a matrix.

Prototype

```
void rad_macos_f(
                  float         *A,
                  rad_stride     ldA,
                  float         *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \cos^{-1}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

rad_vasin_f

Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a vector.

Prototype

```
void rad_vasin_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] := \sin^{-1}($A$[j *$ strideA$])$ where $0 \leqslant j < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

rad_masin_f

Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a matrix.

Prototype

```
void rad_masin_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sin^{-1}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must lie in the interval $[-1, 1]$.

Errors

Notes

rad_vatan_f

Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a vector.

Prototype

```
void rad_vatan_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] := \tan^{-1}($A$[j *$ strideA$])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_matan_f

Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a matrix.

Prototype

```
void rad_matan_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan^{-1}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vatan2_f

Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input vectors.

Prototype

```
void rad_vatan2_f(
                  float         *A,
                  rad_stride     strideA,
                  float         *B,
                  rad_stride     strideB,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \tan^{-1}(\mathtt{A}[j * \mathtt{strideA}]/\mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.
The rules for calculating the function value are the same as those for the ANSI C function `atan2`.

Restrictions

The arguments must not be both zero.

Errors

Notes

rad_matan2_f

Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input matrices.

Prototype

```
void rad_matan2_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *B,
                    rad_stride     ldB,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan^{-1}(\mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$. The rules for calculating the function value are the same as those for the ANSI C function `atan2`.

Restrictions

The arguments must not be both zero.

Errors

Notes

rad_vcos_f

Computes the cosine for each element of a vector. Element angle values are in radians.

Prototype

```
void rad_vcos_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \cos(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

rad_mcos_f

Computes the cosine for each element of a matrix. Element angle values are in radians.

Prototype

```
void rad_mcos_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \cos(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

rad_vcosh_f

Computes the hyperbolic cosine for each element of a vector.

Prototype

```
void rad_vcosh_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \cosh(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mcosh_f

Computes the hyperbolic cosine for each element of a matrix.

Prototype

```
void rad_mcosh_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \cosh(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vceil_f

Computes the ceiling for each element of a vector.

Prototype

```
void rad_vceil_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \lceil \mathtt{A}[j * \mathtt{strideA}] \rceil$ where $0 \leqslant j < n$. Returns the smallest integer greater than or equal to the argument.

Restrictions

Errors

Notes

rad_vexp_f

Computes the exponential function value for each element of a vector.

Prototype

```
void rad_vexp_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \exp(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Overflow will occur if an element is greater than the natural logarithm of the largest representable number.
Underflow will occur if an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

rad_cvexp_f

Computes the exponential function value for each element of a vector.

Prototype

```
void rad_cvexp_f(
                    void          *A,
                    rad_stride     strideA,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \exp(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.
For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.
Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

rad_cvexp_split_f

Computes the exponential function value for each element of a vector.

Prototype

```
void rad_cvexp_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \exp(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.
For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.
Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

rad_mexp_f

Computes the exponential function value for each element of a matrix.

Prototype

```
void rad_mexp_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \exp(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow will occur if an element is greater than the natural logarithm of the largest representable number.
Underflow will occur if an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

rad_cmexp_f

Computes the exponential function value for each element of a matrix.

Prototype

```
void rad_cmexp_f(
                    void          *A,
                    rad_stride     ldA,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \exp(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.
Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

rad_cmexp_split_f

Computes the exponential function value for each element of a matrix.

Prototype

```
void rad_cmexp_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \exp(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
For a complex value $z = x + \mathrm{i}y$ we have $\exp(z) = \exp(x)(\cos(y) + \mathrm{i} \cdot \sin(y))$.

Restrictions

Overflow will occur if the real part of an element is greater than the natural logarithm of the largest representable number.
Underflow will occur if the real part of an element is less than the negative of the natural logarithm of the largest representable number.

Errors

Notes

rad_vexp10_f

Computes the base 10 exponential for each element of a vector.

Prototype

```
void rad_vexp10_f(
                  float         *A,
                  rad_stride     strideA,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := 10^{\mathtt{A}[j*\mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

Overflow will occur if an element is greater than the base 10 logarithm of the largest representable number. Underflow will occur if an element is less than the negative of the base 10 logarithm of the largest representable number.

Errors

Notes

rad_mexp10_f

Computes the base 10 exponential for each element of a matrix.

Prototype

```
void rad_mexp10_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := 10^{\mathtt{A}[j*\mathtt{ldA}+k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow will occur if an element is greater than the base 10 logarithm of the largest representable number. Underflow will occur if an element is less than the negative of the base 10 logarithm of the largest representable number.

Errors

Notes

rad_vfloor_f

Computes the floor for each element of a vector.

Prototype

```
void rad_vfloor_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \lfloor \mathtt{A}[j * \mathtt{strideA}] \rfloor$ where $0 \leqslant j < n$. Returns the largest integer less than or equal to the argument.

Restrictions

Errors

Notes

rad_vlog_f

Computes the natural logarithm for each element of a vector.

Prototype

```
void rad_vlog_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \log_e(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

rad_cvlog_f

Computes the natural logarithm for each element of a vector.

Prototype

```
void rad_cvlog_f(
                    void          *A,
                    rad_stride     strideA,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \log_e(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.
For a complex value z where $0 \leqslant j < n$. we have $\log_e(z) = \log_e(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

rad_cvlog_split_f

Computes the natural logarithm for each element of a vector.

Prototype

```
void rad_cvlog_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \log_e(\texttt{A}[j * \texttt{strideA}])$ where $0 \leqslant j < n$.
For a complex value z where $0 \leqslant j < n$. we have $\log_e(z) = \log_e(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

rad_mlog_f

Computes the natural logarithm for each element of a matrix.

Prototype

```
void rad_mlog_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_e(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

rad_cmlog_f

Computes the natural logarithm for each element of a matrix.

Prototype

```
void rad_cmlog_f(
                    void          *A,
                    rad_stride     ldA,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_e(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
For a complex value z where $0 \leqslant j < m$ and $0 \leqslant k < n$. we have $\log_e(z) = \log_e(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < m$ and $0 \leqslant k < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

rad_cmlog_split_f

Computes the natural logarithm for each element of a matrix.

Prototype

```
void rad_cmlog_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_e(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
For a complex value z where $0 \leqslant j < m$ and $0 \leqslant k < n$. we have $\log_e(z) = \log_e(|z|) + \mathrm{i} \cdot \arg(z)$ where $0 \leqslant j < m$ and $0 \leqslant k < n$..

Restrictions

The arguments must have real and imaginary parts non-zero.

Errors

Notes

rad_vlog10_f

Compute the base ten logarithm for each element of a vector.

Prototype

```
void rad_vlog10_f(
                  float         *A,
                  rad_stride     strideA,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \log_{10}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

The arguments must be greater than zero.

Errors

Notes

rad_mlog10_f

Compute the base ten logarithm for each element of a matrix.

Prototype

```
void rad_mlog10_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \log_{10}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must be greater than zero.

Errors

Notes

rad_vsin_f

Compute the sine for each element of a vector. Element angle values are in radians.

Prototype

```
void rad_vsin_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sin(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

rad_msin_f

Compute the sine for each element of a matrix. Element angle values are in radians.

Prototype

```
void rad_msin_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sin(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Accuracy is decreased for values larger than 8192.

Errors

Notes

Input arguments are expressed in radians.

rad_vsinh_f

Computes the hyperbolic sine for each element of a vector.

Prototype

```
void rad_vsinh_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sinh(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_msinh_f

Computes the hyperbolic sine for each element of a matrix.

Prototype

```
void rad_msinh_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sinh(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvsqrt_P

Compute the square root for each element of a vector.

Prototype

```
void rad_Dvsqrt_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

`rad_vsqrt_f`

`rad_cvsqrt_f`

Parameters

`A`, real or complex vector, length n, input.

`strideA`, integer scalar, input.

`R`, real or complex vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{R}[j * \texttt{strideR}] := \sqrt{\texttt{A}[j * \texttt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

The arguments must be greater than or equal to zero.

Errors

Notes

rad_cvsqrt_split_f

Compute the square root for each element of a vector.

Prototype

```
void rad_cvsqrt_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sqrt{\mathtt{A}[j * \mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmsqrt_P

Compute the square root for each element of a matrix.

Prototype

```
void rad_Dmsqrt_P(
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

`rad_msqrt_f`

`rad_cmsqrt_f`

Parameters

`A`, real or complex matrix, size m by n, input.

`ldA`, integer scalar, input.

`R`, real or complex matrix, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{R}[j * \texttt{ldR} + k] := \sqrt{\texttt{A}[j * \texttt{ldA} + k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The arguments must be greater than or equal to zero.

Errors

Notes

rad_cmsqrt_split_f

Compute the square root for each element of a matrix.

Prototype

```
void rad_cmsqrt_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sqrt{\mathtt{A}[j * \mathtt{ldA} + k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vtan_f

Compute the tangent for each element of a vector. Element angle values are in radians.

Prototype

```
void rad_vtan_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \tan(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

For element values $(n + 1/2)\pi$, the tangent function has a singularity and is undefined.

Errors

Notes

rad_mtan_f

Compute the tangent for each element of a matrix. Element angle values are in radians.

Prototype

```
void rad_mtan_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

For element values $(n + 1/2)\pi$, the tangent function has a singularity and is undefined.

Errors

Notes

rad_vtanh_f

Computes the hyperbolic tangent for each element of a vector.

Prototype

```
void rad_vtanh_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$]$:= tanh(A$[j *$ strideA$]$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mtanh_f

Computes the hyperbolic tangent for each element of a matrix.

Prototype

```
void rad_mtanh_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tanh(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

6.2 Unary Operations

rad_varg_f

rad_marg_f

rad_cvconj_f

rad_cvconj_split_f

rad_cmconj_f

rad_cmconj_split_f

rad_Dvcumsum_P

rad_Dvcumsum_split_P

rad_Dmcumsum_P

rad_Dmcumsum_split_P

rad_veuler_f

rad_meuler_f

rad_Dvmag_P

rad_cvmag_split_f

rad_Dmmag_P

rad_cmmag_split_f

rad_vcmagsq_f

rad_vcmagsq_split_f

rad_mcmagsq_f

rad_Dvmeanval_P

rad_cvmeanval_split_f

rad_Dmmeanval_P

rad_cmmeanval_split_f

rad_Dvmeansqval_P

rad_cvmeansqval_split_f

rad_Dmmeansqval_P

rad_cmmeansqval_split_f

rad_Dvmodulate_P

rad_cvmodulate_split_f

rad_Dvneg_P

rad_cvneg_split_f

rad_Dmneg_P

rad_cmneg_split_f

rad_Dvrecip_P

rad_cvrecip_split_f

rad_Dmrecip_P

rad_cmrecip_split_f

rad_vrsqrt_f

rad_mrsqrt_f

rad_vsq_f

rad_msq_f

rad_Dvsumval_P

rad_Dvsumval_split_P

rad_Dmsumval_P

rad_Dmsumval_split_P

rad_vsumval_bl

rad_vsumsqval_f

rad_msumsqval_f

rad_varg_f

Computes the argument in radians $[-\pi, \pi]$ for each element of a complex vector.

Prototype

```
void rad_varg_f(
                    void          *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ tan^{-1}(imag(A$[j *$ strideA$]$)/real(A$[j *$ strideA$]$)).

Restrictions

Errors

Notes

This function is based on rad_Satan2_f.

rad_marg_f

Computes the argument in radians $[-\pi, \pi]$ for each element of a complex matrix.

Prototype

```
void rad_marg_f(
                    void          *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \tan^{-1}(\text{imag}(\mathtt{A}[j * \mathtt{ldA} + k])/\text{real}(\mathtt{A}[j * \mathtt{ldA} + k]))$.

Restrictions

Errors

Notes

This function is based on rad_Satan2_f.

rad_cvconj_f

Compute the conjugate for each element of a complex vector.

Prototype

```
void rad_cvconj_f(
                    void          *A,
                    rad_stride     strideA,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvconj_split_f

Compute the conjugate for each element of a complex vector.

Prototype

```
void rad_cvconj_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \texttt{A}[j * \texttt{strideA}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cmconj_f

Compute the conjugate for each element of a complex matrix.

Prototype

```
void rad_cmconj_f(
                    void          *A,
                    rad_stride     ldA,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmconj_split_f

Compute the conjugate for each element of a complex matrix.

Prototype

```
void rad_cmconj_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvcumsum_P

Compute the cumulative sum of the elements of a vector.

Prototype

```
void rad_Dvcumsum_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vcumsum_f

rad_vcumsum_i

rad_cvcumsum_f

rad_cvcumsum_i

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j] := \sum_{i=0}^{j} \mathtt{A}[i]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

rad_Dvcumsum_split_P

Compute the cumulative sum of the elements of a vector.

Prototype

```
void rad_Dvcumsum_split_P(
                    rad_scalar_P *A_re,
                    rad_scalar_P *A_im,
                    rad_stride    strideA,
                    rad_scalar_P *R_re,
                    rad_scalar_P *R_im,
                    rad_stride    strideR,
                    unsigned int  n);
```

The following instances are supported:

rad_cvcumsum_split_f

rad_cvcumsum_split_i

Parameters

A_re, real part of real or complex vector, length n, input.

A_im, imaginary part of real or complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of real or complex vector, length n, output.

R_im, imaginary part of real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j] := \sum_{i=0}^{j} \mathtt{A}[i]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

rad_Dmcumsum_P

Compute the cumulative sums of the elements in the rows or columns of a matrix.

Prototype

```
void rad_Dmcumsum_P (
                     rad_major           dir,
                     rad_scalar_P/void  *R,
                     rad_stride          ldR,
                     unsigned int        m,
                     unsigned int        n);
```

The following instances are supported:

rad_mcumsum_f

rad_mcumsum_i

rad_cmcumsum_f

rad_cmcumsum_i

Parameters

dir, enumerated type, input.

RAD_ROW	apply operation to the rows
RAD_COL	apply operation to the columns

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

Row:
$\mathtt{R}[j,k] := \sum_{i=0}^{k} \mathtt{A}[j,i]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Column:
$\mathtt{R}[j,k] := \sum_{i=0}^{j} \mathtt{A}[i,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow may occur.

Errors

Notes

rad_Dmcumsum_split_P

Compute the cumulative sums of the elements in the rows or columns of a matrix.

Prototype

```
void rad_Dmcumsum_split_P (
                    rad_major      dir,
                    rad_scalar_P  *R_re,
                    rad_scalar_P  *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

The following instances are supported:

`rad_cmcumsum_split_f`

`rad_cmcumsum_split_i`

Parameters

`dir`, enumerated type, input.

`RAD_ROW`	apply operation to the rows
`RAD_COL`	apply operation to the columns

`R_re`, real part of real or complex matrix, size m by n, output.

`R_im`, imaginary part of real or complex matrix, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

Row:
$\mathtt{R}[j,k] := \sum_{i=0}^{k} \mathtt{A}[j,i]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Column:
$\mathtt{R}[j,k] := \sum_{i=0}^{j} \mathtt{A}[i,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow may occur.

Errors

Notes

rad_veuler_f

Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a vector.

Prototype

```
void rad_veuler_f(
                  float         *A,
                  rad_stride     strideA,
                  void          *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \cos(\mathtt{A}[j * \mathtt{strideA}]) + \mathrm{i} \cdot \sin(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The speed may be adversely affected for large arguments because of conversion of the argument to its principal value.

rad_meuler_f

Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a matrix.

Prototype

```
void rad_meuler_f(
                  float         *A,
                  rad_stride     ldA,
                  void          *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j*\mathtt{ldR}+k] := \cos(\mathtt{A}[j*\mathtt{ldA}+k]) + \mathrm{i}\cdot\sin(\mathtt{A}[j*\mathtt{ldA}+k]))$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

The speed may be adversely affected for large arguments because of conversion of the argument to its principal value.

rad_Dvmag_P

Compute the magnitude for each element of a vector.

Prototype

```
void rad_Dvmag_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_scalar_P       *R,
                   rad_stride          strideR,
                   unsigned int        n);
```

The following instances are supported:

rad_vmag_f

rad_vmag_i

rad_vmag_si

rad_cvmag_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

R, vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvmag_split_f

Compute the magnitude for each element of a vector.

Prototype

```
void rad_cvmag_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := |\texttt{A}[j * \texttt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmmag_P

Compute the magnitude for each element of a matrix.

Prototype

```
void rad_Dmmag_P (
                    rad_scalar_P/void  *A,
                    rad_stride          ldA,
                    rad_scalar_P       *R,
                    rad_stride          ldR,
                    unsigned int        m,
                    unsigned int        n);
```

The following instances are supported:

`rad_mmag_f`

`rad_cmmag_f`

Parameters

`A`, real or complex matrix, size m by n, input.

`ldA`, integer scalar, input.

`R`, matrix, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`ldR`$ + k] := |$`A`$[j * $`ldA`$ + k]|$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmmag_split_f

Compute the magnitude for each element of a matrix.

Prototype

```
void rad_cmmag_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vcmagsq_f

Computes the square of the magnitudes for each element of a vector.

Prototype

```
void rad_vcmagsq_f(
                   void          *A,
                   rad_stride     strideA,
                   float         *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|^2$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcmagsq_split_f

Computes the square of the magnitudes for each element of a vector.

Prototype

```
void rad_vcmagsq_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := |\texttt{A}[j * \texttt{strideA}]|^2$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mcmagsq_f

Computes the square of the magnitudes for each element of a matrix.

Prototype

```
void rad_mcmagsq_f(
                   void          *A,
                   rad_stride     ldA,
                   float         *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvmeanval_P

Returns the mean value of the elements of a vector.

Prototype

```
rad_Dscalar_P rad_Dvmeanval_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  unsigned int        n);
```

The following instances are supported:

rad_vmeanval_f

rad_cvmeanval_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

real or complex scalar.

Description

return value $:= 1/N \sum$ A$[j *$ strideA$]$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_cvmeanval_split_f

Returns the mean value of the elements of a vector.

Prototype

```
rad_cscalar_f rad_cvmeanval_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

complex scalar.

Description

return value $:= 1/N \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_Dmmeanval_P

Returns the mean value of the elements of a matrix.

Prototype

```
rad_Dscalar_P rad_Dmmeanval_P(
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

rad_mmeanval_f

rad_cmmeanval_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

real or complex scalar.

Description

return value $:= 1/N \sum$ A$[j *$ ldA$+k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_cmmeanval_split_f

Returns the mean value of the elements of a matrix.

Prototype

```
rad_cscalar_f rad_cmmeanval_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

complex scalar.

Description

return value $:= 1/N \sum \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_Dvmeansqval_P

Returns the mean magnitude squared value of the elements of a vector.

Prototype

```
rad_scalar_P rad_Dvmeansqval_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  unsigned int        n);
```

The following instances are supported:

`rad_vmeansqval_f`

`rad_cvmeansqval_f`

Parameters

`A`, real or complex vector, length n, input.

`strideA`, integer scalar, input.

`n`, integer scalar, input.

Return Value

scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j*\mathtt{strideA}]|^2$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_cvmeansqval_split_f

Returns the mean magnitude squared value of the elements of a vector.

Prototype

```
float  rad_cvmeansqval_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j*\mathtt{strideA}]|^2$ where $0 \leqslant j < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_Dmmeansqval_P

Returns the mean magnitude squared value of the elements of a matrix.

Prototype

```
rad_scalar_P rad_Dmmeansqval_P(
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

rad_mmeansqval_f

rad_cmmeansqval_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{ldA} + k]|^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_cmmeansqval_split_f

Returns the mean magnitude squared value of the elements of a matrix.

Prototype

```
float  rad_cmmeansqval_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

real scalar.

Description

return value $:= 1/N \sum |\mathtt{A}[j * \mathtt{ldA} + k]|^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$ where N is the number of elements.

Restrictions

Errors

Notes

rad_Dvmodulate_P

Computes the modulation of a real vector by a specified complex frequency.

Prototype

```
rad_scalar_P rad_Dvmodulate_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P        nu,
                  rad_scalar_P        phi,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vmodulate_f

rad_cvmodulate_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

nu, scalar, input.

phi, scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

scalar.

Description

$\mathtt{R}[j] := \mathtt{A}[j] \cdot (\cos(i \cdot \mathtt{nu} + \mathtt{phi}) + \mathrm{i} \cdot \sin(i \cdot \mathtt{nu} + \mathtt{phi}))$ where $0 \leqslant j < n$,
nu is the frequency in radians per index, and phi is the initial phase.
The function returns $N \cdot \mathtt{nu} + \mathtt{phi}$ where N is the length of the vector. This can be used as the initial phase in the nect call to provide a continuous modulation.

Restrictions

Errors

Notes

To provide continuous filtering but processed by frames the return value can be used as the initial phase for the next frame.

rad_cvmodulate_split_f

Computes the modulation of a real vector by a specified complex frequency.

Prototype

```
float  rad_cvmodulate_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float          nu,
                    float          phi,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

nu, real scalar, input.

phi, real scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

real scalar.

Description

$\texttt{R}[j] := \texttt{A}[j] \cdot (\cos(i \cdot \texttt{nu} + \texttt{phi}) + \mathrm{i} \cdot \sin(i \cdot \texttt{nu} + \texttt{phi}))$ where $0 \leqslant j < n$,
nu is the frequency in radians per index, and phi is the initial phase.
The function returns $N \cdot \texttt{nu} + \texttt{phi}$ where N is the length of the vector. This can be used as the initial phase in the nect call to provide a continuous modulation.

Restrictions

Errors

Notes

To provide continuous filtering but processed by frames the return value can be used as the initial phase for the next frame.

rad_Dvneg_P

Computes the negation for each element of a vector.

Prototype

```
void rad_Dvneg_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_scalar_P/void  *R,
                   rad_stride          strideR,
                   unsigned int        n);
```

The following instances are supported:

- rad_vneg_f
- rad_vneg_i
- rad_vneg_si
- rad_cvneg_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := -\mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvneg_split_f

Computes the negation for each element of a vector.

Prototype

```
void rad_cvneg_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := -\texttt{A}[j * \texttt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmneg_P

Computes the negation for each element of a matrix.

Prototype

```
void rad_Dmneg_P (
                   rad_scalar_P/void  *A,
                   rad_stride          ldA,
                   rad_scalar_P/void  *R,
                   rad_stride          ldR,
                   unsigned int        m,
                   unsigned int        n);
```

The following instances are supported:

rad_mneg_f

rad_mneg_i

rad_cmneg_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := -\mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmneg_split_f

Computes the negation for each element of a matrix.

Prototype

```
void rad_cmneg_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k] := -\texttt{A}[j * \texttt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvrecip_P

Computes the reciprocal for each element of a vector.

Prototype

```
void rad_Dvrecip_P (
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vrecip_f

rad_cvrecip_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := 1/\mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

rad_cvrecip_split_f

Computes the reciprocal for each element of a vector.

Prototype

```
void rad_cvrecip_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := 1/\texttt{A}[j * \texttt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

rad_Dmrecip_P

Computes the reciprocal for each element of a matrix.

Prototype

```
void rad_Dmrecip_P (
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

`rad_mrecip_f`

`rad_cmrecip_f`

Parameters

`A`, real or complex matrix, size m by n, input.

`ldA`, integer scalar, input.

`R`, real or complex matrix, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`ldR`$ + k] := 1/$`A`$[j * $`ldA`$ + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

rad_cmrecip_split_f

Computes the reciprocal for each element of a matrix.

Prototype

```
void rad_cmrecip_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k] := 1/\texttt{A}[j * \texttt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

The divisors must not be zero.

Errors

Notes

rad_vrsqrt_f

Computes the reciprocal of the square root for each element of a vector.

Prototype

```
void rad_vrsqrt_f(
                  float         *A,
                  rad_stride     strideA,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := 1/\sqrt{\mathtt{A}[j * \mathtt{strideA}]}$ where $0 \leqslant j < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

rad_mrsqrt_f

Computes the reciprocal of the square root for each element of a matrix.

Prototype

```
void rad_mrsqrt_f(
                  float         *A,
                  rad_stride     ldA,
                  float         *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := 1/\sqrt{\mathtt{A}[j * \mathtt{ldA} + k]}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Arguments must be greater than zero.

Errors

Notes

rad_vsq_f

Computes the square for each element of a vector.

Prototype

```
void rad_vsq_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]^2$ where $0 \leqslant j < n$.

Restrictions

Overflow will occur if an element's magnitude is greater than the square root of the largest representable number. Underflow will occur if an element's magnitude is less than the square root of the minimum smallest representable number.

Errors

Notes

rad_msq_f

Computes the square for each element of a matrix.

Prototype

```
void rad_msq_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow will occur if an element's magnitude is greater than the square root of the largest representable number. Underflow will occur if an element's magnitude is less than the square root of the minimum smallest representable number.

Errors

Notes

rad_Dvsumval_P

Returns the sum of the elements of a vector.

Prototype

```
rad_scalar_P rad_Dvsumval_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

The following instances are supported:

rad_vsumval_f

rad_vsumval_i

rad_vsumval_si

rad_cvsumval_f

rad_cvsumval_i

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

scalar.

Description

return value := $\sum$ A$[j * \texttt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

rad_Dvsumval_split_P

Returns the sum of the elements of a vector.

Prototype

```
rad_scalar_P rad_Dvsumval_split_P(
                    rad_scalar_P  *A_re,
                    rad_scalar_P  *A_im,
                    rad_stride     strideA,
                    unsigned int   n);
```

The following instances are supported:

rad_cvsumval_split_f

rad_cvsumval_split_i

Parameters

A_re, real part of vector, length n, input.

A_im, imaginary part of vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

scalar.

Description

return value $:= \sum \mathrm{A}[j * \mathrm{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

rad_Dmsumval_P

Returns the sum of the elements of a vector.

Prototype

```
rad_scalar_P rad_Dmsumval_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

The following instances are supported:

rad_msumval_f

rad_msumval_i

rad_cmsumval_f

rad_cmsumval_i

Parameters

A, matrix, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

rad_Dmsumval_split_P

Returns the sum of the elements of a vector.

Prototype

```
rad_scalar_P rad_Dmsumval_split_P(
                    rad_scalar_P  *A_re,
                    rad_scalar_P  *A_im,
                    rad_stride     strideA,
                    unsigned int   n);
```

The following instances are supported:

rad_cmsumval_split_f

rad_cmsumval_split_i

Parameters

A_re, real part of matrix, length n, input.

A_im, imaginary part of matrix, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

rad_vsumval_bl

Returns the sum of the elements of a vector.

Prototype

```
signed int  rad_vsumval_bl(
                    signed int    *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

boolean scalar.

Description

return value := $\sum$ A$[j *$ strideA$]$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

A boolean value is defined so that zero is false and anything else is true. This function returns the number of non-false values in a boolean object. The return type is an unsigned integer type large enough to represent the size of a vector.

rad_vsumsqval_f

Returns the sum of the squares of the elements of a vector.

Prototype

```
float  rad_vsumsqval_f(
                  float         *A,
                  rad_stride     strideA,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

real scalar.

Description

return value $:= \sum \mathtt{A}[j * \mathtt{strideA}]^2$ where $0 \leqslant j < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

rad_msumsqval_f

Returns the sum of the squares of the elements of a matrix.

Prototype

```
float  rad_msumsqval_f(
                  float         *A,
                  rad_stride     ldA,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\sum \mathtt{A}[j * \mathtt{ldA} + k]^2$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Overflow may occur.

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

6.3 Binary Operations

rad_Dvadd_P

rad_Dvadd_split_P

rad_Dmadd_P

rad_cmadd_split_f

rad_rcvadd_f

rad_rcvadd_split_f

rad_rcmadd_f

rad_rcmadd_split_f

rad_Dsvadd_P

rad_csvadd_split_f

rad_Dsmadd_P

rad_csmadd_split_f

rad_rscvadd_f

rad_rscvadd_split_f

rad_rscmadd_f

rad_rscmadd_split_f

rad_Dvdiv_P

rad_Dvdiv_split_P

rad_Dmdiv_P

rad_cmdiv_split_f

rad_rcvdiv_f

rad_rcvdiv_split_f

rad_rcmdiv_f

rad_rcmdiv_split_f

rad_crvdiv_f

rad_crmdiv_f

rad_crmdiv_split_f

rad_rscmsub_f

rad_rscmsub_split_f

rad_Dvsdiv_P

rad_cvrsdiv_split_f

rad_Dmsdiv_P

rad_cmrsdiv_split_f

rad_Dvexpoavg_P

rad_cvexpoavg_split_f

rad_Dmexpoavg_P

rad_cmexpoavg_split_f

rad_vhypot_f

rad_mhypot_f

rad_cvjmul_f

rad_cvjmul_split_f

rad_cmjmul_f

rad_cmjmul_split_f

rad_Dvmul_P

rad_cvmul_split_f

rad_Dmmul_P

rad_cmmul_split_f

rad_rcvmul_f

rad_rcvmul_split_f

rad_rcmmul_f

rad_rcmmul_split_f

rad_rscvmul_f

rad_rscvmul_split_f

rad_csvmul_f

rad_csvmul_split_f

rad_smmul_f

rad_rscmmul_f

rad_rscmmul_split_f

rad_DvDmmul_P

rad_cvmmul_split_f

rad_rvcmmul_f

rad_rvcmmul_split_f

rad_Dvsub_P

rad_cvsub_split_f

rad_Dmsub_P

rad_cmsub_split_f

rad_crmsub_f

rad_crmsub_split_f

rad_rcvsub_f

rad_rcvsub_split_f

rad_rcmsub_f

rad_rcmsub_split_f

rad_crvsub_f

rad_crvsub_split_f

rad_Dsvsub_P

rad_csvsub_split_f

rad_Dsmsub_P

rad_csmsub_split_f

rad_Dsmdiv_P

rad_csmdiv_split_f

rad_rscvdiv_f

rad_rscvdiv_split_f

rad_rscvsub_f

rad_rscvsub_split_f

rad_rscmdiv_f

rad_rscmdiv_split_f

rad_Dvadd_P

Computes the sum, by element, of two vectors.

Prototype

```
void rad_Dvadd_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_scalar_P/void  *B,
                   rad_stride          strideB,
                   rad_scalar_P/void  *R,
                   rad_stride          strideR,
                   unsigned int        n);
```

The following instances are supported:

rad_vadd_f

rad_vadd_i

rad_vadd_si

rad_cvadd_f

rad_cvadd_i

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvadd_split_P

Computes the sum, by element, of two vectors.

Prototype

```
void rad_Dvadd_split_P(
                    rad_scalar_P *A_re,
                    rad_scalar_P *A_im,
                    rad_stride    strideA,
                    rad_scalar_P *B_re,
                    rad_scalar_P *B_im,
                    rad_stride    strideB,
                    rad_scalar_P *R_re,
                    rad_scalar_P *R_im,
                    rad_stride    strideR,
                    unsigned int  n);
```

The following instances are supported:

rad_cvadd_split_f

rad_cvadd_split_i

Parameters

A_re, real part of real or complex vector, length n, input.

A_im, imaginary part of real or complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of real or complex vector, length n, input.

B_im, imaginary part of real or complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of real or complex vector, length n, output.

R_im, imaginary part of real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \text{strideR}] :=$ A$[j * \text{strideA}] +$ B$[j * \text{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmadd_P

Computes the sum, by element, of two matrices.

Prototype

```
void rad_Dmadd_P (
                    rad_scalar_P/void  *A,
                    rad_stride          ldA,
                    rad_scalar_P/void  *B,
                    rad_stride          ldB,
                    rad_scalar_P/void  *R,
                    rad_stride          ldR,
                    unsigned int        m,
                    unsigned int        n);
```

The following instances are supported:

rad_madd_f

rad_madd_i

rad_cmadd_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmadd_split_f

Computes the sum, by element, of two matrices.

Prototype

```
void rad_cmadd_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rcvadd_f

Computes the sum, by element, of two vectors.

Prototype

```
void rad_rcvadd_f(
                    float         *A,
                    rad_stride     strideA,
                    void          *B,
                    rad_stride     strideB,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rcvadd_split_f

Computes the sum, by element, of two vectors.

Prototype

```
void rad_rcvadd_split_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rcmadd_f

Computes the sum, by element, of two matrices.

Prototype

```
void rad_rcmadd_f(
                    float         *A,
                    rad_stride     ldA,
                    void          *B,
                    rad_stride     ldB,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rcmadd_split_f

Computes the sum, by element, of two matrices.

Prototype

```
void rad_rcmadd_split_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dsvadd_P

Computes the sum, by element, of a scalar and a vector.

Prototype

```
void rad_Dsvadd_P(
                    rad_Dscalar_P        a,
                    rad_scalar_P/void   *B,
                    rad_stride           strideB,
                    rad_scalar_P/void   *R,
                    rad_stride           strideR,
                    unsigned int         n);
```

The following instances are supported:

rad_svadd_f

rad_svadd_i

rad_svadd_si

rad_csvadd_f

Parameters

a, real or complex scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ a $+$ B$[j *$ strideB$]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_csvadd_split_f

Computes the sum, by element, of a scalar and a vector.

Prototype

```
void rad_csvadd_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ a $+$ B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dsmadd_P

Computes the sum, by element, of a scalar and a matrix.

Prototype

```
void rad_Dsmadd_P(
                  rad_Dscalar_P       a,
                  rad_scalar_P/void  *B,
                  rad_stride          ldB,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

rad_smadd_f

rad_smadd_i

rad_csmadd_f

Parameters

a, real or complex scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_csmadd_split_f

Computes the sum, by element, of a scalar and a matrix.

Prototype

```
void rad_csmadd_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k] :=$ a $+$ B$[j * \texttt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rscvadd_f

Computes the sum, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvadd_f(
                   float          a,
                   void          *B,
                   rad_stride     strideB,
                   void          *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ a $+$ B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rscvadd_split_f

Computes the sum, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvadd_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ a $+$ B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rscmadd_f

Computes the sum, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmadd_f(
                   float          a,
                   void          *B,
                   rad_stride     ldB,
                   void          *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} + \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rscmadd_split_f

Computes the sum, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmadd_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k] :=$ a $+$ B$[j * \texttt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvdiv_P

Computes the quotient, by element, of two vectors.

Prototype

```
void rad_Dvdiv_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vdiv_f

rad_cvdiv_f

rad_svdiv_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_Dvdiv_split_P

Computes the quotient, by element, of two vectors.

Prototype

```
void rad_Dvdiv_split_P(
                    rad_scalar_P *A_re,
                    rad_scalar_P *A_im,
                    rad_stride    strideA,
                    rad_scalar_P *B_re,
                    rad_scalar_P *B_im,
                    rad_stride    strideB,
                    rad_scalar_P *R_re,
                    rad_scalar_P *R_im,
                    rad_stride    strideR,
                    unsigned int  n);
```

The following instances are supported:

rad_crvdiv_split_f

rad_cvdiv_split_f

Parameters

A_re, real part of real or complex vector, length n, input.

A_im, imaginary part of real or complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of real or complex vector, length n, input.

B_im, imaginary part of real or complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of real or complex vector, length n, output.

R_im, imaginary part of real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_Dmdiv_P

Computes the quotient, by element, of two matrices.

Prototype

```
void rad_Dmdiv_P (
                   rad_scalar_P/void  *A,
                   rad_stride          ldA,
                   rad_scalar_P/void  *B,
                   rad_stride          ldB,
                   rad_scalar_P/void  *R,
                   rad_stride          ldR,
                   unsigned int        m,
                   unsigned int        n);
```

The following instances are supported:

rad_mdiv_f

rad_cmdiv_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] / \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_cmdiv_split_f

Computes the quotient, by element, of two matrices.

Prototype

```
void rad_cmdiv_split_f(
                  float         *A_re,
                  float         *A_im,
                  rad_stride     ldA,
                  float         *B_re,
                  float         *B_im,
                  rad_stride     ldB,
                  float         *R_re,
                  float         *R_im,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] / \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rcvdiv_f

Computes the quotient, by element, of two vectors.

Prototype

```
void rad_rcvdiv_f(
                  float          a,
                  void          *B,
                  rad_stride     strideB,
                  void          *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ a$/$B$[j *$ strideB$]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rcvdiv_split_f

Computes the quotient, by element, of two vectors.

Prototype

```
void rad_rcvdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a}/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rcmdiv_f

Computes the quotient, by element, of two matrices.

Prototype

```
void rad_rcmdiv_f(
                    float          a,
                    void          *B,
                    rad_stride     ldB,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rcmdiv_split_f

Computes the quotient, by element, of two matrices.

Prototype

```
void rad_rcmdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_crvdiv_f

Computes the quotient, by element, of two vectors.

Prototype

```
void rad_crvdiv_f(
                    void          *A,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_crmdiv_f

Computes the quotient, by element, of two matrices.

Prototype

```
void rad_crmdiv_f(
                    void          *A,
                    rad_stride     ldA,
                    float         *B,
                    rad_stride     ldB,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] / \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_crmdiv_split_f

Computes the quotient, by element, of two matrices.

Prototype

```
void rad_crmdiv_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *B,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] / \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rscmsub_f

Computes the difference, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmsub_f(
                   float          a,
                   void          *B,
                   rad_stride     ldB,
                   void          *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k] :=$ a $-$ B$[j * \texttt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rscmsub_split_f

Computes the difference, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmsub_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvsdiv_P

Computes the quotient, by element, of a vector and a scalar.

Prototype

```
void rad_Dvsdiv_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_Dscalar_P       b,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vsdiv_f

rad_cvrsdiv_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

b, real or complex scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ A$[j * \texttt{strideA}]/$b where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

rad_cvrsdiv_split_f

Computes the quotient, by element, of a vector and a scalar.

Prototype

```
void rad_cvrsdiv_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float          b_re,
                    float          b_im,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of real vector, length n, input.

A_im, imaginary part of real vector, length n, input.

strideA, integer scalar, input.

b_re, real part of real scalar, input.

b_im, imaginary part of real scalar, input.

R_re, real part of real vector, length n, output.

R_im, imaginary part of real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ A$[j *$ strideA$]/$b where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

rad_Dmsdiv_P

Computes the quotient, by element, of a matrix and a scalar.

Prototype

```
void rad_Dmsdiv_P(
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  rad_Dscalar_P       b,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

rad_msdiv_f

rad_cmrsdiv_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

b, real or complex scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{b}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

rad_cmrsdiv_split_f

Computes the quotient, by element, of a matrix and a scalar.

Prototype

```
void rad_cmrsdiv_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *b_re,
                    float         *b_im,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

b_re, real part of complex scalar, input.

b_im, imaginary part of complex scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]/\mathtt{b}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

rad_Dvexpoavg_P

Computes an exponential weighted average, by element, of two vectors.

Prototype

```
void rad_Dvexpoavg_P (
                  rad_scalar_P         a,
                  rad_scalar_P/void   *B,
                  rad_stride           strideB,
                  rad_scalar_P/void   *C,
                  rad_stride           strideC,
                  unsigned int         n);
```

The following instances are supported:

rad_vexpoavg_f

rad_cvexpoavg_f

Parameters

a, scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

C, real or complex vector, length n, modified in place.

strideC, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$C[j * strideC] := a \cdot B[j * strideB] + (1 - a) \cdot C[j * strideC]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvexpoavg_split_f

Computes an exponential weighted average, by element, of two vectors.

Prototype

```
void rad_cvexpoavg_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *C_re,
                    float         *C_im,
                    rad_stride     strideC,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

C_re, real part of complex vector, length n, modified in place.

C_im, imaginary part of complex vector, length n, modified in place.

strideC, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{C}[j * \mathtt{strideC}] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{strideB}] + (1 - \mathtt{a}) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmexpoavg_P

Computes an exponential weighted average, by element, of two matrices.

Prototype

```
void rad_Dmexpoavg_P (
                  rad_scalar_P         a,
                  rad_scalar_P/void   *B,
                  rad_stride           ldB,
                  rad_scalar_P/void   *C,
                  rad_stride           ldC,
                  unsigned int         m,
                  unsigned int         n);
```

The following instances are supported:

`rad_mexpoavg_f`

`rad_cmexpoavg_f`

Parameters

`a`, scalar, input.

`B`, real or complex matrix, size m by n, input.

`ldB`, integer scalar, input.

`C`, real or complex matrix, size m by n, modified in place.

`ldC`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`C`$[j*$`ldC`$+k] :=$ `a`$\cdot$`B`$[j*$`ldB`$+k]+(1-$`a`$)\cdot$`C`$[j*$`ldC`$+k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmexpoavg_split_f

Computes an exponential weighted average, by element, of two matrices.

Prototype

```
void rad_cmexpoavg_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *C_re,
                    float         *C_im,
                    rad_stride     ldC,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

C_re, real part of complex matrix, size m by n, modified in place.

C_im, imaginary part of complex matrix, size m by n, modified in place.

ldC, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{C}[j*\mathtt{ldC}+k] := \mathtt{a}\cdot\mathtt{B}[j*\mathtt{ldB}+k]+(1-\mathtt{a})\cdot\mathtt{C}[j*\mathtt{ldC}+k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vhypot_f

Computes the square root of the sum of squares, by element, of two input vectors.

Prototype

```
void rad_vhypot_f(
                  float         *A,
                  rad_stride     strideA,
                  float         *B,
                  rad_stride     strideB,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \sqrt{(\mathtt{A}[j * \mathtt{strideA}])^2 + (\mathtt{B}[j * \mathtt{strideB}])^2}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

Intermediate overflows do not occur.

rad_mhypot_f

Computes the square root of the sum of squares, by element, of two input matrices.

Prototype

```
void rad_mhypot_f(
                  float         *A,
                  rad_stride     ldA,
                  float         *B,
                  rad_stride     ldB,
                  float         *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \sqrt{(\mathtt{A}[j * \mathtt{ldA} + k])^2 + (\mathtt{B}[j * \mathtt{ldB} + k])^2}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

Intermediate overflows do not occur.

rad_cvjmul_f

Computes the product of a complex vector with the conjugate of a second complex vector, by element.

Prototype

```
void rad_cvjmul_f(
                  void          *A,
                  rad_stride     strideA,
                  void          *B,
                  rad_stride     strideB,
                  void          *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvjmul_split_f

Computes the product of a complex vector with the conjugate of a second complex vector, by element.

Prototype

```
void rad_cvjmul_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \text{strideR}] := \text{A}[j * \text{strideA}] \cdot \text{B}[j * \text{strideB}]^*$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cmjmul_f

Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.

Prototype

```
void rad_cmjmul_f(
                    void          *A,
                    rad_stride     ldA,
                    void          *B,
                    rad_stride     ldB,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmjmul_split_f

Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.

Prototype

```
void rad_cmjmul_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]^*$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvmul_P

Computes the product, by element, of two vectors.

Prototype

```
void rad_Dvmul_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vmul_f

rad_vmul_i

rad_vmul_si

rad_cvmul_f

rad_svmul_f

rad_svmul_i

rad_svmul_si

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvmul_split_f

Computes the product, by element, of two vectors.

Prototype

```
void rad_cvmul_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ A$[j * \texttt{strideA}] \cdot$ B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmmul_P

Computes the product, by element, of two matrices.

Prototype

```
void rad_Dmmul_P (
                   rad_scalar_P/void  *A,
                   rad_stride          ldA,
                   rad_scalar_P/void  *B,
                   rad_stride          ldB,
                   rad_scalar_P/void  *R,
                   rad_stride          ldR,
                   unsigned int        m,
                   unsigned int        n);
```

The following instances are supported:

- rad_mmul_f
- rad_mmul_i
- rad_cmmul_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmmul_split_f

Computes the product, by element, of two matrices.

Prototype

```
void rad_cmmul_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rcvmul_f

Computes the product, by element, of two vectors.

Prototype

```
void rad_rcvmul_f(
                    float         *A,
                    rad_stride     strideA,
                    void          *B,
                    rad_stride     strideB,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rcvmul_split_f

Computes the product, by element, of two vectors.

Prototype

```
void rad_rcvmul_split_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rcmmul_f

Computes the product, by element, of two matrices.

Prototype

```
void rad_rcmmul_f(
                  float         *A,
                  rad_stride     ldA,
                  void          *B,
                  rad_stride     ldB,
                  void          *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rcmmul_split_f

Computes the product, by element, of two matrices.

Prototype

```
void rad_rcmmul_split_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rscvmul_f

Computes the product, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvmul_f(
                   float          a,
                   void          *B,
                   rad_stride     strideB,
                   void          *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \texttt{a} \cdot \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rscvmul_split_f

Computes the product, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvmul_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \texttt{a} \cdot \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_csvmul_f

Computes the product, by element, of a scalar and a vector.

Prototype

```
void rad_csvmul_f(
                    void          *a,
                    void          *B,
                    rad_stride     strideB,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a, complex scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \texttt{a} \cdot \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_csvmul_split_f

Computes the product, by element, of a scalar and a vector.

Prototype

```
void rad_csvmul_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \texttt{a} \cdot \texttt{B}[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_smmul_f

Computes the product, by element, of a scalar and a matrix.

Prototype

```
void rad_smmul_f(
                    float          a,
                    float         *B,
                    rad_stride     ldB,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rscmmul_f

Computes the product, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmmul_f(
                   float          a,
                   void          *B,
                   rad_stride     ldB,
                   void          *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rscmmul_split_f

Computes the product, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmmul_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_DvDmmul_P

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void rad_DvDmmul_P (
                    rad_scalar_P/void  *A,
                    rad_stride          strideA,
                    rad_scalar_P/void  *B,
                    rad_stride          ldB,
                    rad_major           major,
                    rad_scalar_P/void  *R,
                    rad_stride          ldR,
                    unsigned int        m,
                    unsigned int        n);
```

The following instances are supported:

`rad_vmmul_f`

`rad_cvmmul_f`

Parameters

`A`, real or complex vector, input. Length n when by rows; length m when by columns.

`strideA`, integer scalar, input.

`B`, real or complex matrix, size m by n, input.

`ldB`, integer scalar, input.

`major`, enumerated type, input.

`RAD_ROW`	apply operation to the rows
`RAD_COL`	apply operation to the columns

`R`, real or complex matrix, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions
Errors

The arguments must conform to the following:

1. `major` must be valid.

Notes

rad_cvmmul_split_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void rad_cvmmul_split_f(
                    float/void   *A_re,
                    float/void   *A_im,
                    rad_stride    strideA,
                    float/void   *B_re,
                    float/void   *B_im,
                    rad_stride    ldB,
                    rad_major     major,
                    float/void   *R_re,
                    float/void   *R_im,
                    rad_stride    ldR,
                    unsigned int  m,
                    unsigned int  n);
```

Parameters

A_re, real part of real or complex vector, length p, input.

A_im, imaginary part of real or complex vector, length p, input. Length n when by rows; length m when by columns.

strideA, integer scalar, input.

B_re, real part of real or complex matrix, size m by n, input.

B_im, imaginary part of real or complex matrix, size m by n, input.

ldB, integer scalar, input.

major, enumerated type, input.

RAD_ROW	apply operation to the rows
RAD_COL	apply operation to the columns

R_re, real part of real or complex matrix, size m by n, output.

R_im, imaginary part of real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions
Errors

The arguments must conform to the following:

1. major must be valid.

Notes

rad_rvcmmul_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void rad_rvcmmul_f (
                    float         *A,
                    rad_stride     strideA,
                    void          *B,
                    rad_stride     ldB,
                    rad_major      major,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real vector, input. Length n when by rows; length m when by columns.

strideA, integer scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

major, enumerated type, input.

RAD_ROW	apply operation to the rows
RAD_COL	apply operation to the columns

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

By rows: $R[j,k] := A[k] \cdot B[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $R[j,k] := A[j] \cdot B[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions
Errors

The arguments must conform to the following:

1. major must be valid.

Notes

rad_rvcmmul_split_f

Computes the product, by element, of a vector and the rows or columns of a matrix.

Prototype

```
void rad_rvcmmul_split_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    rad_major      major,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real vector, input. Length n when by rows; length m when by columns.

strideA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

major, enumerated type, input.

RAD_ROW	apply operation to the rows
RAD_COL	apply operation to the columns

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

By rows: $\mathtt{R}[j,k] := \mathtt{A}[k] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.
By columns: $\mathtt{R}[j,k] := \mathtt{A}[j] \cdot \mathtt{B}[j,k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions
Errors

The arguments must conform to the following:

1. major must be valid.

Notes

rad_Dvsub_P

Computes the difference, by element, of two vectors.

Prototype

```
void rad_Dvsub_P (
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vsub_f

rad_vsub_i

rad_vsub_si

rad_cvsub_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvsub_split_f

Computes the difference, by element, of two vectors.

Prototype

```
void rad_cvsub_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ A$[j *$ strideA$] -$ B$[j *$ strideB$]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmsub_P

Computes the difference, by element, of two matrices.

Prototype

```
void rad_Dmsub_P (
                   rad_scalar_P/void  *A,
                   rad_stride          ldA,
                   rad_scalar_P/void  *B,
                   rad_stride          ldB,
                   rad_scalar_P/void  *R,
                   rad_stride          ldR,
                   unsigned int        m,
                   unsigned int        n);
```

The following instances are supported:

- rad_msub_f
- rad_msub_i
- rad_cmsub_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmsub_split_f

Computes the difference, by element, of two matrices.

Prototype

```
void rad_cmsub_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_crmsub_f

Computes the difference, by element, of two matrices.

Prototype

```
void rad_crmsub_f(
                  void          *A,
                  rad_stride     ldA,
                  float         *B,
                  rad_stride     ldB,
                  void          *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_crmsub_split_f

Computes the difference, by element, of two matrices.

Prototype

```
void rad_crmsub_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *B,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rcvsub_f

Computes the difference, by element, of two vectors.

Prototype

```
void rad_rcvsub_f(
                    float         *A,
                    rad_stride     strideA,
                    void          *B,
                    rad_stride     strideB,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rcvsub_split_f

Computes the difference, by element, of two vectors.

Prototype

```
void rad_rcvsub_split_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rcmsub_f

Computes the difference, by element, of two matrices.

Prototype

```
void rad_rcmsub_f(
                  float         *A,
                  rad_stride     ldA,
                  void          *B,
                  rad_stride     ldB,
                  void          *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_rcmsub_split_f

Computes the difference, by element, of two matrices.

Prototype

```
void rad_rcmsub_split_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_crvsub_f

Computes the difference, by element, of two vectors.

Prototype

```
void rad_crvsub_f(
                    void          *A,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_crvsub_split_f

Computes the difference, by element, of two vectors.

Prototype

```
void rad_crvsub_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dsvsub_P

Computes the difference, by element, of a scalar and a vector.

Prototype

```
void rad_Dsvsub_P(
                  rad_Dscalar_P       a,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_svsub_f

rad_svsub_i

rad_svsub_si

rad_csvsub_f

Parameters

a, real or complex scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$R[j * strideR] := a - B[j * strideB]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_csvsub_split_f

Computes the difference, by element, of a scalar and a vector.

Prototype

```
void rad_csvsub_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a} - \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dsmsub_P

Computes the difference, by element, of a scalar and a matrix.

Prototype

```
void rad_Dsmsub_P(
                  rad_Dscalar_P       a,
                  rad_scalar_P/void  *B,
                  rad_stride          ldB,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

rad_smsub_f

rad_smsub_i

rad_csmsub_f

Parameters

a, real or complex scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a} - \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_csmsub_split_f

Computes the difference, by element, of a scalar and a matrix.

Prototype

```
void rad_csmsub_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k] :=$ a $-$ B$[j * \texttt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dsmdiv_P

Computes the quotient, by element, of a scalar and a matrix.

Prototype

```
void rad_Dsmdiv_P(
                  rad_Dscalar_P       a,
                  rad_scalar_P/void  *B,
                  rad_stride          ldB,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

`rad_smdiv_f`

`rad_csmdiv_f`

Parameters

`a`, real or complex scalar, input.

`B`, real or complex matrix, size m by n, input.

`ldB`, integer scalar, input.

`R`, real or complex matrix, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`ldR`$ + k] :=$ `a`$/$`B`$[j * $`ldB`$ + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

rad_csmdiv_split_f

Computes the quotient, by element, of a scalar and a matrix.

Prototype

```
void rad_csmdiv_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k]$:= a/B$[j * \texttt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero.

Errors

Notes

rad_rscvdiv_f

Computes the quotient, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvdiv_f(
                   float          a,
                   void          *B,
                   rad_stride     strideB,
                   void          *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ a/B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rscvdiv_split_f

Computes the quotient, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{a}/\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rscvsub_f

Computes the difference, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvsub_f(
                   float          a,
                   void          *B,
                   rad_stride     strideB,
                   void          *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ a $-$ B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rscvsub_split_f

Computes the difference, by element, of a real scalar and a complex vector.

Prototype

```
void rad_rscvsub_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ a $-$ B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_rscmdiv_f

Computes the quotient, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmdiv_f(
                   float          a,
                   void          *B,
                   rad_stride     ldB,
                   void          *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n);
```

Parameters

a, real scalar, input.

B, complex matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

rad_rscmdiv_split_f

Computes the quotient, by element, of a real scalar and a complex matrix.

Prototype

```
void rad_rscmdiv_split_f(
                    float          a,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a, real scalar, input.

B_re, real part of complex matrix, size m by n, input.

B_im, imaginary part of complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}/\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Divisors must not be zero. Overflows and underflows are possible.

Errors

Notes

6.4 Ternary Operations

rad_Dvam_P

rad_Dvam_split_P

rad_Dvmsa_P

rad_cvmsa_split_f

rad_Dvmsb_P

rad_cvmsb_split_f

rad_Dvsam_P

rad_cvsam_split_f

rad_Dvsbm_P

rad_cvsbm_split_f

rad_Dvsma_P

rad_cvsma_split_f

rad_Dvsmsa_P

rad_cvsmsa_split_f

rad_Dvam_P

Computes the sum of two vectors and product of a third vector, by element.

Prototype

```
void rad_Dvam_P (
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  rad_scalar_P/void  *C,
                  rad_stride          strideC,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vam_f

rad_vma_f

rad_cvam_f

rad_cvma_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

C, real or complex vector, length n, input.

strideC, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvam_split_P

Computes the sum of two vectors and product of a third vector, by element.

Prototype

```
void rad_Dvam_split_P(
                  rad_scalar_P *A_re,
                  rad_scalar_P *A_im,
                  rad_stride    strideA,
                  rad_scalar_P *B_re,
                  rad_scalar_P *B_im,
                  rad_stride    strideB,
                  rad_scalar_P *C_re,
                  rad_scalar_P *C_im,
                  rad_stride    strideC,
                  rad_scalar_P *R_re,
                  rad_scalar_P *R_im,
                  rad_stride    strideR,
                  unsigned int  n);
```

The following instances are supported:

rad_cvam_split_f

rad_cvma_split_f

Parameters

A_re, real part of real or complex vector, length n, input.

A_im, imaginary part of real or complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of real or complex vector, length n, input.

B_im, imaginary part of real or complex vector, length n, input.

strideB, integer scalar, input.

C_re, real part of real or complex vector, length n, input.

C_im, imaginary part of real or complex vector, length n, input.

strideC, integer scalar, input.

R_re, real part of real or complex vector, length n, output.

R_im, imaginary part of real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] + \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvmsa_P

Computes the product of two vectors and sum of a scalar, by element.

Prototype

```
void rad_Dvmsa_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_scalar_P/void  *B,
                   rad_stride          strideB,
                   rad_Dscalar_P       c,
                   rad_scalar_P/void  *R,
                   rad_stride          strideR,
                   unsigned int        n);
```

The following instances are supported:

`rad_vmsa_f`

`rad_cvmsa_f`

Parameters

`A`, real or complex vector, length n, input.

`strideA`, integer scalar, input.

`B`, real or complex vector, length n, input.

`strideB`, integer scalar, input.

`c`, real or complex scalar, input.

`R`, real or complex vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] :=$ `A`$[j * $`strideA`$] \cdot$ `B`$[j * $`strideB`$] +$ `c` where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvmsa_split_f

Computes the product of two vectors and sum of a scalar, by element.

Prototype

```
void rad_cvmsa_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *c_re,
                    float         *c_im,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

c_re, real part of complex scalar, input.

c_im, imaginary part of complex scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}] + \mathtt{c}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvmsb_P

Computes the product of two vectors and difference of a third vector, by element.

Prototype

```
void rad_Dvmsb_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_scalar_P/void  *B,
                   rad_stride          strideB,
                   rad_scalar_P/void  *C,
                   rad_stride          strideC,
                   rad_scalar_P/void  *R,
                   rad_stride          strideR,
                   unsigned int        n);
```

The following instances are supported:

`rad_vmsb_f`

`rad_cvmsb_f`

Parameters

`A`, real or complex vector, length n, input.

`strideA`, integer scalar, input.

`B`, real or complex vector, length n, input.

`strideB`, integer scalar, input.

`C`, real or complex vector, length n, input.

`strideC`, integer scalar, input.

`R`, real or complex vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] :=$ `A`$[j * $`strideA`$] \cdot$ `B`$[j * $`strideB`$] -$ `C`$[j * $`strideC`$]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvmsb_split_f

Computes the product of two vectors and difference of a third vector, by element.

Prototype

```
void rad_cvmsb_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *C_re,
                    float         *C_im,
                    rad_stride     strideC,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

C_re, real part of complex vector, length n, input.

C_im, imaginary part of complex vector, length n, input.

strideC, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{B}[j * \mathtt{strideB}] - \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvsam_P

Computes the sum of a vector and a scalar, and product with a second vector, by element.

Prototype

```
void rad_Dvsam_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_Dscalar_P       b,
                   rad_scalar_P/void  *C,
                   rad_stride          strideC,
                   rad_scalar_P/void  *R,
                   rad_stride          strideR,
                   unsigned int        n);
```

The following instances are supported:

rad_vsam_f

rad_cvsam_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

b, real or complex scalar, input.

C, real or complex vector, length n, input.

strideC, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ (A$[j *$ strideA$] +$ b$) \cdot$ C$[j *$ strideC$]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvsam_split_f

Computes the sum of a vector and a scalar, and product with a second vector, by element.

Prototype

```
void rad_cvsam_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *b_re,
                    float         *b_im,
                    float         *C_re,
                    float         *C_im,
                    rad_stride     strideC,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

b_re, real part of complex scalar, input.

b_im, imaginary part of complex scalar, input.

C_re, real part of complex vector, length n, input.

C_im, imaginary part of complex vector, length n, input.

strideC, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ (A$[j *$ strideA$]$ + b$) \cdot$ C$[j *$ strideC$]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvsbm_P

Computes the difference of two vectors, and product with a third vector, by element.

Prototype

```
void rad_Dvsbm_P (
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  rad_scalar_P/void  *C,
                  rad_stride          strideC,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vsbm_f

rad_cvsbm_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

B, real or complex vector, length n, input.

strideB, integer scalar, input.

C, real or complex vector, length n, input.

strideC, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvsbm_split_f

Computes the difference of two vectors, and product with a third vector, by element.

Prototype

```
void rad_cvsbm_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *C_re,
                    float         *C_im,
                    rad_stride     strideC,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

C_re, real part of complex vector, length n, input.

C_im, imaginary part of complex vector, length n, input.

strideC, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] - \mathtt{B}[j * \mathtt{strideB}]) \cdot \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvsma_P

Computes the product of a vector and a scalar, and sum with a second vector, by element.

Prototype

```
void rad_Dvsma_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_Dscalar_P       b,
                   rad_scalar_P/void  *C,
                   rad_stride          strideC,
                   rad_scalar_P/void  *R,
                   rad_stride          strideR,
                   unsigned int        n);
```

The following instances are supported:

rad_vsma_f

rad_cvsma_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

b, real or complex scalar, input.

C, real or complex vector, length n, input.

strideC, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$] :=$ A$[j *$ strideA$] \cdot$ b $+$ C$[j *$ strideC$]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvsma_split_f

Computes the product of a vector and a scalar, and sum with a second vector, by element.

Prototype

```
void rad_cvsma_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *b_re,
                    float         *b_im,
                    float         *C_re,
                    float         *C_im,
                    rad_stride     strideC,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

b_re, real part of complex scalar, input.

b_im, imaginary part of complex scalar, input.

C_re, real part of complex vector, length n, input.

C_im, imaginary part of complex vector, length n, input.

strideC, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] \cdot \mathtt{b} + \mathtt{C}[j * \mathtt{strideC}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dvsmsa_P

Computes the product of a vector and a scalar, and sum with a second scalar, by element.

Prototype

```
void rad_Dvsmsa_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_Dscalar_P       b,
                  rad_Dscalar_P       c,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vsmsa_f

rad_cvsmsa_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

b, real or complex scalar, input.

c, real or complex scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \text{strideR}] :=$ A$[j * \text{strideA}] \cdot$ b $+$ c where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvsmsa_split_f

Computes the product of a vector and a scalar, and sum with a second scalar, by element.

Prototype

```
void rad_cvsmsa_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *b_re,
                    float         *b_im,
                    float         *c_re,
                    float         *c_im,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

b_re, real part of complex scalar, input.

b_im, imaginary part of complex scalar, input.

c_re, real part of complex scalar, input.

c_im, imaginary part of complex scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \text{strideR}] :=$ A$[j * \text{strideA}] \cdot$ b $+$ c where $0 \leqslant j < n$.

Restrictions

Errors

Notes

6.5 Logical Operations

rad_valltrue_bl

rad_malltrue_bl

rad_vanytrue_bl

rad_manytrue_bl

rad_Dvleq_P

rad_Dvleq_split_P

rad_Dmleq_P

rad_Dmleq_split_P

rad_vlge_P

rad_mlge_P

rad_vlgt_P

rad_mlgt_P

rad_vlle_P

rad_mlle_P

rad_vllt_P

rad_mllt_P

rad_Dvlne_P

rad_Dvlne_split_P

rad_Dmlne_P

rad_Dmlne_split_P

rad_valltrue_bl

Returns true if all the elements of a vector are true.

Prototype

```
signed int  rad_valltrue_bl(
                    signed int    *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

boolean scalar.

Description

return value := (for all elements $\mathtt{A}[j * \mathtt{strideA}] = \mathtt{true}$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_malltrue_bl

Returns true if all the elements of a vector are true.

Prototype

```
signed int  rad_malltrue_bl(
                    signed int    *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

boolean scalar.

Description

return value := (for all elements $\mathtt{A}[j * \mathtt{strideA}] = \mathtt{true}$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vanytrue_bl

Returns true if one or more elements of a vector are true.

Prototype

```
signed int  rad_vanytrue_bl(
                    signed int    *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

boolean scalar.

Description

return value := (for any element $\mathtt{A}[j * \mathtt{strideA}] = \mathtt{true}$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The logical complement of any true is none true.

rad_manytrue_bl

Returns true if one or more elements of a vector are true.

Prototype

```
signed int  rad_manytrue_bl(
                    signed int    *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

boolean scalar.

Description

return value := (for any element $\texttt{A}[j * \texttt{strideA}] = \texttt{true}$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The logical complement of any true is none true.

rad_Dvleq_P

Computes the boolean comparison of ‘equal’, by element, of two vectors.

Prototype

```
void rad_Dvleq_P (
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  signed int         *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

`rad_vleq_f`

`rad_vleq_i`

`rad_cvleq_f`

`rad_cvleq_i`

Parameters

`A`, real or complex vector, length n, input.

`strideA`, integer scalar, input.

`B`, real or complex vector, length n, input.

`strideB`, integer scalar, input.

`R`, boolean vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{strideA}] = \texttt{B}[j * \texttt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_Dvleq_split_P

Computes the boolean comparison of ‘equal’, by element, of two vectors.

Prototype

```
void rad_Dvleq_split_P (
                  rad_scalar_P *A_re,
                  rad_scalar_P *A_im,
                  rad_stride    strideA,
                  rad_scalar_P *B_re,
                  rad_scalar_P *B_im,
                  rad_stride    strideB,
                  signed int   *R_re,
                  signed int   *R_im,
                  rad_stride    strideR,
                  unsigned int  n);
```

The following instances are supported:

rad_cvleq_split_f

rad_cvleq_split_i

Parameters

A_re, real part of real or complex vector, length n, input.

A_im, imaginary part of real or complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of real or complex vector, length n, input.

B_im, imaginary part of real or complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of boolean vector, length n, output.

R_im, imaginary part of boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] = \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_Dmleq_P

Computes the boolean comparison of 'equal', by element, of two vectors/matrices.

Prototype

```
void rad_Dmleq_P (
                   rad_scalar_P/void  *A,
                   rad_stride          ldA,
                   rad_scalar_P/void  *B,
                   rad_stride          ldB,
                   signed int         *R,
                   rad_stride          strideR,
                   unsigned int        m,
                   unsigned int        n);
```

The following instances are supported:

- rad_mleq_f
- rad_mleq_i
- rad_cmleq_f
- rad_cmleq_i

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] = \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_Dmleq_split_P

Computes the boolean comparison of 'equal', by element, of two vectors/matrices.

Prototype

```
void rad_Dmleq_split_P(
                  rad_scalar_P *A_re,
                  rad_scalar_P *A_im,
                  rad_stride    ldA,
                  rad_scalar_P *B_re,
                  rad_scalar_P *B_im,
                  rad_stride    ldB,
                  signed int   *R_re,
                  signed int   *R_im,
                  rad_stride    strideR,
                  unsigned int  m,
                  unsigned int  n);
```

The following instances are supported:

rad_cmleq_split_f

rad_cmleq_split_i

Parameters

A_re, real part of real or complex matrix, size m by n, input.

A_im, imaginary part of real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of real or complex matrix, size m by n, input.

B_im, imaginary part of real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of boolean vector, length n, output.

R_im, imaginary part of boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] = \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_vlge_P

Computes the boolean comparison of ‘greater than or equal’, by element, of two vectors.

Prototype

```
void rad_vlge_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    rad_scalar_P  *B,
                    rad_stride     strideB,
                    signed int    *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

The following instances are supported:

rad_vlge_f

rad_vlge_i

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

B, vector, length n, input.

strideB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] \geqslant \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_mlge_P

Computes the boolean comparison of 'greater than or equal', by element, of two vectors/matrices.

Prototype

```
void rad_mlge_P (
                  rad_scalar_P  *A,
                  rad_stride     ldA,
                  rad_scalar_P  *B,
                  rad_stride     ldB,
                  signed int    *R,
                  rad_stride     strideR,
                  unsigned int   m,
                  unsigned int   n);
```

The following instances are supported:

rad_mlge_f

rad_mlge_i

Parameters

A, matrix, size m by n, input.

ldA, integer scalar, input.

B, matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] \geqslant \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_vlgt_P

Computes the boolean comparison of ‘greater than’, by element, of two vectors.

Prototype

```
void rad_vlgt_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    rad_scalar_P  *B,
                    rad_stride     strideB,
                    signed int    *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

The following instances are supported:

rad_vlgt_f

rad_vlgt_i

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

B, vector, length n, input.

strideB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] > \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_mlgt_P

Computes the boolean comparison of 'greater than', by element, of two vectors/matrices.

Prototype

```
void rad_mlgt_P (
                  rad_scalar_P  *A,
                  rad_stride     ldA,
                  rad_scalar_P  *B,
                  rad_stride     ldB,
                  signed int    *R,
                  rad_stride     strideR,
                  unsigned int   m,
                  unsigned int   n);
```

The following instances are supported:

rad_mlgt_f

rad_mlgt_i

Parameters

A, matrix, size m by n, input.

ldA, integer scalar, input.

B, matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] > \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_vlle_P

Computes the boolean comparison of ‘less than or equal’, by element, of two vectors.

Prototype

```
void rad_vlle_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    rad_scalar_P  *B,
                    rad_stride     strideB,
                    signed int    *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

The following instances are supported:

rad_vlle_f

rad_vlle_i

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

B, vector, length n, input.

strideB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] \leqslant \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_mlle_P

Computes the boolean comparison of 'less than or equal', by element, of two vectors/matrices.

Prototype

```
void rad_mlle_P(
                    rad_scalar_P  *A,
                    rad_stride     ldA,
                    rad_scalar_P  *B,
                    rad_stride     ldB,
                    signed int    *R,
                    rad_stride     strideR,
                    unsigned int   m,
                    unsigned int   n);
```

The following instances are supported:

rad_mlle_f

rad_mlle_i

Parameters

A, matrix, size m by n, input.

ldA, integer scalar, input.

B, matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] \leqslant \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_vllt_P

Computes the boolean comparison of 'less than', by element, of two vectors.

Prototype

```
void rad_vllt_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    rad_scalar_P  *B,
                    rad_stride     strideB,
                    signed int    *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

The following instances are supported:

`rad_vllt_f`

`rad_vllt_i`

Parameters

`A`, vector, length n, input.

`strideA`, integer scalar, input.

`B`, vector, length n, input.

`strideB`, integer scalar, input.

`R`, boolean vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`strideR`$] :=$ (`A`$[j * $`strideA`$] <$ `B`$[j * $`strideB`$])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_mllt_P

Computes the boolean comparison of 'less than', by element, of two vectors/matrices.

Prototype

```
void rad_mllt_P(
                    rad_scalar_P  *A,
                    rad_stride     ldA,
                    rad_scalar_P  *B,
                    rad_stride     ldB,
                    signed int    *R,
                    rad_stride     strideR,
                    unsigned int   m,
                    unsigned int   n);
```

The following instances are supported:

rad_mllt_f

rad_mllt_i

Parameters

A, matrix, size m by n, input.

ldA, integer scalar, input.

B, matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] < \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_Dvlne_P

Computes the boolean comparison of ‘not equal’, by element, of two vectors.

Prototype

```
void rad_Dvlne_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  signed int         *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

`rad_vlne_f`

`rad_vlne_i`

`rad_cvlne_f`

`rad_cvlne_i`

Parameters

`A`, real or complex vector, length n, input.

`strideA`, integer scalar, input.

`B`, real or complex vector, length n, input.

`strideB`, integer scalar, input.

`R`, boolean vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{R}[j * \texttt{strideR}] := (\texttt{A}[j * \texttt{strideA}] \neq \texttt{B}[j * \texttt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_Dvlne_split_P

Computes the boolean comparison of ‘not equal’, by element, of two vectors.

Prototype

```
void rad_Dvlne_split_P (
                  rad_scalar_P *A_re,
                  rad_scalar_P *A_im,
                  rad_stride    strideA,
                  rad_scalar_P *B_re,
                  rad_scalar_P *B_im,
                  rad_stride    strideB,
                  signed int   *R_re,
                  signed int   *R_im,
                  rad_stride    strideR,
                  unsigned int  n);
```

The following instances are supported:

rad_cvlne_split_f

rad_cvlne_split_i

Parameters

A_re, real part of real or complex vector, length n, input.

A_im, imaginary part of real or complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of real or complex vector, length n, input.

B_im, imaginary part of real or complex vector, length n, input.

strideB, integer scalar, input.

R_re, real part of boolean vector, length n, output.

R_im, imaginary part of boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{strideA}] \neq \mathtt{B}[j * \mathtt{strideB}])$ where $0 \leqslant j < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_Dmlne_P

Computes the boolean comparison of 'not equal', by element, of two vectors/matrices.

Prototype

```
void rad_Dmlne_P (
                   rad_scalar_P/void  *A,
                   rad_stride          ldA,
                   rad_scalar_P/void  *B,
                   rad_stride          ldB,
                   signed int         *R,
                   rad_stride          strideR,
                   unsigned int        m,
                   unsigned int        n);
```

The following instances are supported:

rad_mlne_f

rad_mlne_i

rad_cmlne_f

rad_cmlne_i

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B, real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] \neq \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

rad_Dmlne_split_P

Computes the boolean comparison of 'not equal', by element, of two vectors/matrices.

Prototype

```
void rad_Dmlne_split_P (
                  rad_scalar_P *A_re,
                  rad_scalar_P *A_im,
                  rad_stride    ldA,
                  rad_scalar_P *B_re,
                  rad_scalar_P *B_im,
                  rad_stride    ldB,
                  signed int   *R_re,
                  signed int   *R_im,
                  rad_stride    strideR,
                  unsigned int  m,
                  unsigned int  n);
```

The following instances are supported:

rad_cmlne_split_f

rad_cmlne_split_i

Parameters

A_re, real part of real or complex matrix, size m by n, input.

A_im, imaginary part of real or complex matrix, size m by n, input.

ldA, integer scalar, input.

B_re, real part of real or complex matrix, size m by n, input.

B_im, imaginary part of real or complex matrix, size m by n, input.

ldB, integer scalar, input.

R_re, real part of boolean vector, length n, output.

R_im, imaginary part of boolean vector, length n, output.

strideR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := (\mathtt{A}[j * \mathtt{ldA} + k] \neq \mathtt{B}[j * \mathtt{ldB} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Since the input and output vectors are of a different precision there is no in-place functionality for this function.

Errors

Notes

6.6 Selection Operations

rad_vclip_P

rad_vinvclip_P

rad_vindexbool

rad_vmax_f

rad_vmaxmg_f

rad_vcmaxmgsq_f

rad_vcmaxmgsq_split_f

rad_vcmaxmgsqval_f

rad_vcmaxmgsqval_split_f

rad_vmaxmgval_f

rad_vmaxval_f

rad_vmin_f

rad_vminmg_f

rad_vcminmgsq_f

rad_vcminmgsq_split_f

rad_vcminmgsqval_f

rad_vcminmgsqval_split_f

rad_vminmgval_f

rad_vminval_f

rad_vclip_P

Computes the generalised double clip, by element, of two vectors.

Prototype

```
void rad_vclip_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    rad_scalar_P   t1,
                    rad_scalar_P   t2,
                    rad_scalar_P   c1,
                    rad_scalar_P   c2,
                    rad_scalar_P  *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

The following instances are supported:

rad_vclip_f

rad_vclip_i

rad_vclip_si

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

t1, scalar, input.

t2, scalar, input.

c1, scalar, input.

c2, scalar, input.

R, vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

If $\mathtt{A}[j * \mathtt{strideA}] \leqslant \mathtt{t1}$ then $\mathtt{R}[j * \mathtt{strideR}] := \mathtt{c1}$ else if $\mathtt{A}[j * \mathtt{strideA}] < \mathtt{t2}$ then $\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ else $\mathtt{R}[j * \mathtt{strideR}] := \mathtt{c2}$.

Restrictions

Errors

Notes

The clipping rules are evaluated sequentially: once a rule is met, the following rules are ignored. The threshold variables are unrestricted and need not be in increasing order.

rad_vinvclip_P

Computes the generalised inverted double clip, by element, of two vectors.

Prototype

```
void rad_vinvclip_P(
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    rad_scalar_P   t1,
                    rad_scalar_P   t2,
                    rad_scalar_P   t3,
                    rad_scalar_P   c1,
                    rad_scalar_P   c2,
                    rad_scalar_P  *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

The following instances are supported:

rad_vinvclip_f

rad_vinvclip_i

rad_vinvclip_si

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

t1, scalar, input.

t2, scalar, input.

t3, scalar, input.

c1, scalar, input.

c2, scalar, input.

R, vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

If $\mathtt{A}[j*\mathtt{strideA}] < \mathtt{t1}$ then $\mathtt{R}[j*\mathtt{strideR}] := \mathtt{A}[j*\mathtt{strideA}]$ else if $\mathtt{A}[j*\mathtt{strideA}] < \mathtt{t2}$ then $\mathtt{R}[j*\mathtt{strideR}] := \mathtt{c1}$ else if $\mathtt{A}[j*\mathtt{strideA}] \leqslant \mathtt{t3}$ then $\mathtt{R}[j*\mathtt{strideR}] := \mathtt{c2}$ else $\mathtt{R}[j*\mathtt{strideR}] := \mathtt{A}[j*\mathtt{strideA}]$.

Restrictions

Errors

Notes

The clipping rules are evaluated sequentially: once a rule is met, the following rules are ignored. The threshold variables are unrestricted and need not be in increasing order.

rad_vindexbool

Computes an index vector of the indices of the non-false elements of the boolean vector, and returns the number of non-false elements.

Prototype

```
unsigned int  rad_vindexbool(
                    signed int    *X,
                    rad_stride     strideX,
                    unsigned int  *Y,
                    rad_stride     strideY,
                    unsigned int   n);
```

Parameters

X, boolean vector, length n, input.

strideX, integer scalar, input.

Y, vector-index vector, length n, output.

strideY, integer scalar, input.

n, integer scalar, input.

Return Value

integer scalar.

Description

Returns an index vector Y of the indices of the non-false elements of the boolean vector X. The index vector is ordered: lower indices appear before higher indices. The return value is the number of non-false elements.

Restrictions

No in-place operations are allowed.

Errors

Notes

rad_vmax_f

Computes the maximum, by element, of two vectors.

Prototype

```
void rad_vmax_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \max\{\mathtt{A}[j * \mathtt{strideA}], \mathtt{B}[j * \mathtt{strideB}]\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vmaxmg_f

Computes the maximum magnitude (absolute value), by element, of two vectors.

Prototype

```
void rad_vmaxmg_f(
                  float         *A,
                  rad_stride     strideA,
                  float         *B,
                  rad_stride     strideB,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \max\{|\mathtt{A}[j * \mathtt{strideA}]|, |\mathtt{B}[j * \mathtt{strideB}]|\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcmaxmgsq_f

Computes the maximum magnitude squared, by element, of two complex vectors.

Prototype

```
void rad_vcmaxmgsq_f(
                    void          *A,
                    rad_stride     strideA,
                    void          *B,
                    rad_stride     strideB,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \max\{|\mathtt{A}[j * \mathtt{strideA}]|^2, |\mathtt{B}[j * \mathtt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcmaxmgsq_split_f

Computes the maximum magnitude squared, by element, of two complex vectors.

Prototype

```
void rad_vcmaxmgsq_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] := \max\{|\texttt{A}[j * \texttt{strideA}]|^2, |\texttt{B}[j * \texttt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcmaxmgsqval_f

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.

Prototype

```
float  rad_vcmaxmgsqval_f(
                  void           *A,
                  rad_stride      strideA,
                  unsigned int   *index,
                  unsigned int    n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

index, pointer to vector-index scalar, output.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\max\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.
If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum magnitude squared values, the index of the first maximum magnitude squared is returned in the index.

rad_vcmaxmgsqval_split_f

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.

Prototype

```
float  rad_vcmaxmgsqval_split_f(
                   float          *A_re,
                   float          *A_im,
                   rad_stride      strideA,
                   unsigned int   *index,
                   unsigned int    n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

index, pointer to vector-index scalar, output.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\max\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.
If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum magnitude squared values, the index of the first maximum magnitude squared is returned in the index.

rad_vmaxmgval_f

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

```
float  rad_vmaxmgval_f(
                  float          *A,
                  rad_stride      strideA,
                  unsigned int   *index,
                  unsigned int    n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

index, pointer to vector-index scalar, output.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\max\{|A[j * strideA]|\}$ where $0 \leqslant j < n$.
If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum absolute values, the index of the first maximum absolute value is returned in the index.

rad_vmaxval_f

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.

Prototype

```
float  rad_vmaxval_f(
                  float          *A,
                  rad_stride      strideA,
                  unsigned int   *index,
                  unsigned int    n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

index, pointer to vector-index scalar, output.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\max\{\mathtt{A}[j * \mathtt{strideA}]\}$ where $0 \leqslant j < n$.
If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical maximum values the index of the first maximum is returned in the index.

rad_vmin_f

Computes the minimum, by element, of two vectors.

Prototype

```
void rad_vmin_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{\mathtt{A}[j * \mathtt{strideA}], \mathtt{B}[j * \mathtt{strideB}]\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vminmg_f

Computes the minimum magnitude (absolute value), by element, of two vectors.

Prototype

```
void rad_vminmg_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{|\mathtt{A}[j * \mathtt{strideA}]|, |\mathtt{B}[j * \mathtt{strideB}]|\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcminmgsq_f

Computes the minimum magnitude squared, by element, of two complex vectors.

Prototype

```
void rad_vcminmgsq_f(
                    void          *A,
                    rad_stride     strideA,
                    void          *B,
                    rad_stride     strideB,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{|\mathtt{A}[j * \mathtt{strideA}]|^2, |\mathtt{B}[j * \mathtt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcminmgsq_split_f

Computes the minimum magnitude squared, by element, of two complex vectors.

Prototype

```
void rad_vcminmgsq_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \min\{|\mathtt{A}[j * \mathtt{strideA}]|^2, |\mathtt{B}[j * \mathtt{strideB}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcminmgsqval_f

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.

Prototype

```
float  rad_vcminmgsqval_f(
                  void           *A,
                  rad_stride      strideA,
                  unsigned int   *index,
                  unsigned int    n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

index, pointer to vector-index scalar, output.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\min\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum magnitude squared values, the index of the first minimum magnitude squared is returned in the index.

rad_vcminmgsqval_split_f

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.

Prototype

```
float  rad_vcminmgsqval_split_f(
                   float          *A_re,
                   float          *A_im,
                   rad_stride      strideA,
                   unsigned int   *index,
                   unsigned int    n);
```

Parameters

`A_re`, real part of complex vector, length n, input.

`A_im`, imaginary part of complex vector, length n, input.

`strideA`, integer scalar, input.

`index`, pointer to vector-index scalar, output.

`n`, integer scalar, input.

Return Value

real scalar.

Description

return value := $\min\{|\mathtt{A}[j * \mathtt{strideA}]|^2\}$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum magnitude squared values, the index of the first minimum magnitude squared is returned in the index.

rad_vminmgval_f

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.

Prototype

```
float  rad_vminmgval_f(
                  float          *A,
                  rad_stride      strideA,
                  unsigned int   *index,
                  unsigned int    n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

index, pointer to vector-index scalar, output.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\min\{|A[j * strideA]|\}$ where $0 \leqslant j < n$.
If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum absolute value values, the index of the first minimum absolute value is returned in the index.

rad_vminval_f

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.

Prototype

```
float  rad_vminval_f(
                  float          *A,
                  rad_stride      strideA,
                  unsigned int   *index,
                  unsigned int    n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

index, pointer to vector-index scalar, output.

n, integer scalar, input.

Return Value

real scalar.

Description

return value := $\min\{A[j * \texttt{strideA}]\}$ where $0 \leqslant j < n$.
If index is not NULL the index is returned.

Restrictions

Errors

Notes

If the vector has more than one element with identical minimum values the index of the first minimum is returned in the index.

6.7 Bitwise and Boolean Logical Operators

rad_vand_P

rad_mand_P

rad_vand_bl

rad_mand_bl

rad_vnot_P

rad_mnot_P

rad_vnot_bl

rad_mnot_bl

rad_vor_P

rad_mor_P

rad_vor_bl

rad_mor_bl

rad_vxor_P

rad_mxor_P

rad_vxor_bl

rad_mxor_bl

rad_vand_P

Computes the bitwise and, by element, of two vectors.

Prototype

```
void rad_vand_P (
                  rad_scalar_P  *A,
                  rad_stride     strideA,
                  rad_scalar_P  *B,
                  rad_stride     strideB,
                  rad_scalar_P  *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

The following instances are supported:

rad_vand_i

rad_vand_si

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

B, vector, length n, input.

strideB, integer scalar, input.

R, vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j *$ strideR$]$:= A$[j *$ strideA$]$ and B$[j *$ strideB$]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mand_P

Computes the bitwise and, by element, of two matrices.

Prototype

```
void rad_mand_P (
                  rad_scalar_P  *A,
                  rad_stride     ldA,
                  rad_scalar_P  *B,
                  rad_stride     ldB,
                  rad_scalar_P  *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

The following instances are supported:

rad_mand_i

rad_mand_si

Parameters

A, matrix, size m by n, input.

ldA, integer scalar, input.

B, matrix, size m by n, input.

ldB, integer scalar, input.

R, matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ and $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vand_bl

Computes the logical and, by element, of two vectors.

Prototype

```
void rad_vand_bl(
                    signed int   *A,
                    rad_stride    strideA,
                    signed int   *B,
                    rad_stride    strideB,
                    signed int   *R,
                    rad_stride    strideR,
                    unsigned int  n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

B, boolean vector, length n, input.

strideB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ and $\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mand_bl

Computes the logical and, by element, of two matrices.

Prototype

```
void rad_mand_bl(
                    signed int    *A,
                    rad_stride     ldA,
                    signed int    *B,
                    rad_stride     ldB,
                    signed int    *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, boolean matrix, size m by n, input.

ldA, integer scalar, input.

B, boolean matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ and $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vnot_P

Computes the bitwise not (one's complement), by element, of two vectors.

Prototype

```
void rad_vnot_P (
                  rad_scalar_P  *A,
                  rad_stride     strideA,
                  rad_scalar_P  *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

The following instances are supported:

rad_vnot_i

rad_vnot_si

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

R, vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \text{not}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mnot_P

Computes the bitwise not (one's complement), by element, of two matrices.

Prototype

```
void rad_mnot_P (
                  rad_scalar_P  *A,
                  rad_stride     ldA,
                  rad_scalar_P  *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

The following instances are supported:

`rad_mnot_i`

`rad_mnot_si`

Parameters

`A`, matrix, size m by n, input.

`ldA`, integer scalar, input.

`R`, matrix, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j * $`ldR`$ + k] :=$ not(`A`$[j * $`ldA`$ + k]$) where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vnot_bl

Computes the logical not, by element, of two vectors.

Prototype

```
void rad_vnot_bl(
                    signed int    *A,
                    rad_stride     strideA,
                    signed int    *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \text{not}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mnot_bl

Computes the logical not, by element, of two matrices.

Prototype

```
void rad_mnot_bl(
                    signed int   *A,
                    rad_stride    ldA,
                    signed int   *R,
                    rad_stride    ldR,
                    unsigned int  m,
                    unsigned int  n);
```

Parameters

A, boolean matrix, size m by n, input.

ldA, integer scalar, input.

R, boolean matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \text{not}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vor_P

Computes the bitwise inclusive or, by element, of two vectors.

Prototype

```
void rad_vor_P (
                    rad_scalar_P  *A,
                    rad_stride     strideA,
                    rad_scalar_P  *B,
                    rad_stride     strideB,
                    rad_scalar_P  *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

The following instances are supported:

rad_vor_i

rad_vor_si

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

B, vector, length n, input.

strideB, integer scalar, input.

R, vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ A$[j * \texttt{strideA}]$ or B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mor_P

Computes the bitwise inclusive or, by element, of two matrices.

Prototype

```
void rad_mor_P (
                    rad_scalar_P  *A,
                    rad_stride     ldA,
                    rad_scalar_P  *B,
                    rad_stride     ldB,
                    rad_scalar_P  *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

The following instances are supported:

rad_mor_i

rad_mor_si

Parameters

A, matrix, size m by n, input.

ldA, integer scalar, input.

B, matrix, size m by n, input.

ldB, integer scalar, input.

R, matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ or $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vor_bl

Computes the logical inclusive or, by element, of two vectors.

Prototype

```
void rad_vor_bl(
                    signed int   *A,
                    rad_stride    strideA,
                    signed int   *B,
                    rad_stride    strideB,
                    signed int   *R,
                    rad_stride    strideR,
                    unsigned int  n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

B, boolean vector, length n, input.

strideB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ or $\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mor_bl

Computes the logical inclusive or, by element, of two matrices.

Prototype

```
void rad_mor_bl(
                    signed int   *A,
                    rad_stride    ldA,
                    signed int   *B,
                    rad_stride    ldB,
                    signed int   *R,
                    rad_stride    ldR,
                    unsigned int  m,
                    unsigned int  n);
```

Parameters

A, boolean matrix, size m by n, input.

ldA, integer scalar, input.

B, boolean matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ or $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vxor_P

Computes the bitwise exclusive or, by element, of two vectors.

Prototype

```
void rad_vxor_P (
                  rad_scalar_P  *A,
                  rad_stride     strideA,
                  rad_scalar_P  *B,
                  rad_stride     strideB,
                  rad_scalar_P  *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

The following instances are supported:

rad_vxor_i

rad_vxor_si

Parameters

A, vector, length n, input.

strideA, integer scalar, input.

B, vector, length n, input.

strideB, integer scalar, input.

R, vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}]$:= A$[j * \texttt{strideA}]$ xor B$[j * \texttt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mxor_P

Computes the bitwise exclusive or, by element, of two matrices.

Prototype

```
void rad_mxor_P (
                  rad_scalar_P  *A,
                  rad_stride     ldA,
                  rad_scalar_P  *B,
                  rad_stride     ldB,
                  rad_scalar_P  *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

The following instances are supported:

rad_mxor_i

rad_mxor_si

Parameters

A, matrix, size m by n, input.

ldA, integer scalar, input.

B, matrix, size m by n, input.

ldB, integer scalar, input.

R, matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ xor $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vxor_bl

Computes the logical exclusive or, by element, of two vectors.

Prototype

```
void rad_vxor_bl(
                    signed int   *A,
                    rad_stride    strideA,
                    signed int   *B,
                    rad_stride    strideB,
                    signed int   *R,
                    rad_stride    strideR,
                    unsigned int  n);
```

Parameters

A, boolean vector, length n, input.

strideA, integer scalar, input.

B, boolean vector, length n, input.

strideB, integer scalar, input.

R, boolean vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ xor $\mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mxor_bl

Computes the logical exclusive or, by element, of two matrices.

Prototype

```
void rad_mxor_bl(
                    signed int    *A,
                    rad_stride     ldA,
                    signed int    *B,
                    rad_stride     ldB,
                    signed int    *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, boolean matrix, size m by n, input.

ldA, integer scalar, input.

B, boolean matrix, size m by n, input.

ldB, integer scalar, input.

R, boolean matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ xor $\mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

6.8 Element Generation and Copy

rad_Dvcopy_P_P

rad_cvcopy_split_f_f

rad_Dmcopy_P_P

rad_cmcopy_split_f_f

rad_Dvfill_P

rad_cvfill_split_f

rad_Dmfill_P

rad_cmfill_split_f

rad_vramp_P

rad_Dvcopy_P_P

Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void rad_Dvcopy_P_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *R,
                  rad_stride          strideR,
                  unsigned int        n);
```

The following instances are supported:

rad_vcopy_f_f

rad_vcopy_f_i

rad_vcopy_f_si

rad_vcopy_f_bl

rad_vcopy_i_f

rad_vcopy_i_i

rad_vcopy_i_si

rad_vcopy_i_vi

rad_vcopy_si_f

rad_vcopy_si_i

rad_vcopy_si_si

rad_vcopy_bl_f

rad_vcopy_bl_bl

rad_vcopy_vi_i

rad_vcopy_vi_vi

rad_vcopy_mi_mi

rad_cvcopy_f_f

Parameters

A, real or complex vector, length n, input.

strideA, integer scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

`R`$[j * \texttt{strideR}] := $ `A`$[j * \texttt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

rad_cvcopy_split_f_f

Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void rad_cvcopy_split_f_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}]$ where $0 \leqslant j < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

rad_Dmcopy_P_P

Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void rad_Dmcopy_P_P (
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

rad_mcopy_f_f

rad_mcopy_f_i

rad_mcopy_f_si

rad_mcopy_f_bl

rad_mcopy_i_f

rad_mcopy_i_i

rad_mcopy_i_si

rad_mcopy_si_f

rad_mcopy_si_i

rad_mcopy_si_si

rad_mcopy_bl_bl

rad_cmcopy_f_f

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

rad_cmcopy_split_f_f

Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.

Prototype

```
void rad_cmcopy_split_f_f(
                    rad_scalar_P  *A_re,
                    rad_scalar_P  *A_im,
                    rad_stride     ldA,
                    rad_scalar_P  *R_re,
                    rad_scalar_P  *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

If the source and destination overlap, the result is undefined.

Errors

Notes

When copying from a boolean variable, false and true map onto zero and one respectively. When copying to a boolean variable, zero maps to false and everything else maps to true.

rad_Dvfill_P

Fill a vector with a constant value.

Prototype

```
void rad_Dvfill_P(
                  rad_Dscalar_P        a,
                  rad_scalar_P/void   *R,
                  rad_stride           strideR,
                  unsigned int         n);
```

The following instances are supported:

- rad_vfill_f
- rad_vfill_i
- rad_vfill_si
- rad_cvfill_f

Parameters

a, real or complex scalar, input.

R, real or complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$R[j * strideR] := a$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_cvfill_split_f

Fill a vector with a constant value.

Prototype

```
void rad_cvfill_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ a where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_Dmfill_P

Fill a matrix with a constant value.

Prototype

```
void rad_Dmfill_P(
                  rad_Dscalar_P       a,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

rad_mfill_f

rad_mfill_i

rad_mfill_si

rad_cmfill_f

Parameters

a, real or complex scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{a}$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_cmfill_split_f

Fill a matrix with a constant value.

Prototype

```
void rad_cmfill_split_f(
                    float         *a_re,
                    float         *a_im,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

a_re, real part of complex scalar, input.

a_im, imaginary part of complex scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{ldR} + k] :=$ a where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vramp_P

Computes a vector ramp by starting at an initial value and incrementing each successive element by the ramp step size.

Prototype

```
void rad_vramp_P (
                  rad_scalar_P   alpha,
                  rad_scalar_P   beta,
                  rad_scalar_P  *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

The following instances are supported:

`rad_vramp_f`

`rad_vramp_i`

`rad_vramp_si`

Parameters

`alpha`, scalar, input.

`beta`, scalar, input.

`R`, vector, length n, output.

`strideR`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R`$[j] :=$ `alpha` $+ j \cdot$ `beta` where $0 \leqslant j < n$.

Restrictions

Errors

Notes

6.9 Manipulation Operations

rad_vcmplx_f

rad_vcmplx_split_f

rad_mcmplx_f

rad_Dvgather_P

rad_cvgather_split_f

rad_Dmgather_P

rad_cmgather_split_f

rad_vimag_f

rad_vimag_split_f

rad_mimag_f

rad_vpolar_f

rad_vpolar_split_f

rad_mpolar_f

rad_mpolar_split_f

rad_vreal_f

rad_vreal_split_f

rad_mreal_f

rad_vrect_f

rad_vrect_split_f

rad_mrect_f

rad_mrect_split_f

rad_Dvscatter_P

rad_cvscatter_split_f

rad_Dmscatter_P

rad_cmscatter_split_f

rad_Dvswap_P

rad_cvswap_split_f

rad_Dmswap_P

rad_cmswap_split_f

rad_vcmplx_f

Form a complex vector from two real vectors.

Prototype

```
void rad_vcmplx_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    void          *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R, complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathrm{i} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vcmplx_split_f

Form a complex vector from two real vectors.

Prototype

```
void rad_vcmplx_split_f(
                    float         *A,
                    rad_stride     strideA,
                    float         *B,
                    rad_stride     strideB,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

B, real vector, length n, input.

strideB, integer scalar, input.

R_re, real part of complex vector, length n, output.

R_im, imaginary part of complex vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \mathtt{A}[j * \mathtt{strideA}] + \mathrm{i} \cdot \mathtt{B}[j * \mathtt{strideB}]$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mcmplx_f

Form a complex matrix from two real matrices.

Prototype

```
void rad_mcmplx_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *B,
                    rad_stride     ldB,
                    void          *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, real matrix, size m by n, input.

ldA, integer scalar, input.

B, real matrix, size m by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \mathtt{A}[j * \mathtt{ldA} + k] + \mathrm{i} \cdot \mathtt{B}[j * \mathtt{ldB} + k]$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_Dvgather_P

The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void rad_Dvgather_P(
                    rad_scalar_P/void  *X,
                    rad_stride          strideX,
                    unsigned int       *I,
                    rad_stride          strideI,
                    rad_scalar_P/void  *Y,
                    rad_stride          strideY,
                    unsigned int        n);
```

The following instances are supported:

`rad_vgather_f`

`rad_vgather_i`

`rad_vgather_si`

`rad_cvgather_f`

Parameters

`X`, real or complex vector, length m, input.

`strideX`, integer scalar, input.

`I`, vector-index vector, length n, input.

`strideI`, integer scalar, input.

`Y`, real or complex vector, length n, output.

`strideY`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{Y}[j] := \texttt{X}[\texttt{I}[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

rad_cvgather_split_f

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.

Prototype

```
void rad_cvgather_split_f(
                    float          *X_re,
                    float          *X_im,
                    rad_stride      strideX,
                    unsigned int   *I_re,
                    unsigned int   *I_im,
                    rad_stride      strideI,
                    float          *Y_re,
                    float          *Y_im,
                    rad_stride      strideY,
                    unsigned int    n);
```

Parameters

X_re, real part of complex vector, length m, input.

X_im, imaginary part of complex vector, length m, input.

strideX, integer scalar, input.

I_re, real part of vector-index vector, length n, input.

I_im, imaginary part of vector-index vector, length n, input.

strideI, integer scalar, input.

Y_re, real part of complex vector, length n, output.

Y_im, imaginary part of complex vector, length n, output.

strideY, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{Y}[j] := \mathtt{X}[\mathtt{I}[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

rad_Dmgather_P

The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.

Prototype

```
void rad_Dmgather_P(
                    rad_scalar_P/void  *X,
                    rad_stride          ldX,
                    rad_scalar_mi      *I,
                    rad_stride          strideI,
                    rad_scalar_P/void  *Y,
                    rad_stride          strideY,
                    unsigned int        n);
```

The following instances are supported:

`rad_mgather_f`

`rad_mgather_i`

`rad_mgather_si`

`rad_cmgather_f`

Parameters

`X`, real or complex matrix, size m by n, input.

`ldX`, integer scalar, input.

`I`, matrix-index vector, length p, input.

`strideI`, integer scalar, input.

`Y`, real or complex vector, length p, output.

`strideY`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{Y}[j] := \texttt{X}[\texttt{I}[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

rad_cmgather_split_f

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.

Prototype

```
void rad_cmgather_split_f(
                    float           *X_re,
                    float           *X_im,
                    rad_stride       ldX,
                    rad_scalar_mi   *I_re,
                    rad_scalar_mi   *I_im,
                    rad_stride       strideI,
                    float           *Y_re,
                    float           *Y_im,
                    rad_stride       strideY,
                    unsigned int     n);
```

Parameters

X_re, real part of complex matrix, size m by n, input.

X_im, imaginary part of complex matrix, size m by n, input.

ldX, integer scalar, input.

I_re, real part of matrix-index vector, length p, input.

I_im, imaginary part of matrix-index vector, length p, input.

strideI, integer scalar, input.

Y_re, real part of complex vector, length p, output.

Y_im, imaginary part of complex vector, length p, output.

strideY, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{Y}[j] := \mathtt{X}[\mathtt{I}[j]]$ where $0 \leqslant j < n$.

Restrictions

The length of the destination vector must be the same size as the index vector.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the source vector.

Notes

The destination vector must be the same size as the index vector.

rad_vimag_f

Extract the imaginary part of a complex vector.

Prototype

```
void rad_vimag_f(
                    void          *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := \text{imag}(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vimag_split_f

Extract the imaginary part of a complex vector.

Prototype

```
void rad_vimag_split_f(
                   float         *A_re,
                   float         *A_im,
                   rad_stride     strideA,
                   float         *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ imag(A$[j * \texttt{strideA}]$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mimag_f

Extract the imaginary part of a complex matrix.

Prototype

```
void rad_mimag_f(
                    void          *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \text{imag}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vpolar_f

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.

Prototype

```
void rad_vpolar_f(
                  void          *A,
                  rad_stride     strideA,
                  float         *R,
                  rad_stride     strideR,
                  float         *P,
                  rad_stride     strideP,
                  unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

P, real vector, length n, output.

strideP, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ and $\mathtt{P}[j * \mathtt{strideP}] := \arg(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vpolar_split_f

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.

Prototype

```
void rad_vpolar_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    float         *P,
                    rad_stride     strideP,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

P, real vector, length n, output.

strideP, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{strideR}] := |\mathtt{A}[j * \mathtt{strideA}]|$ and $\mathtt{P}[j * \mathtt{strideP}] := \arg(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mpolar_f

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.

Prototype

```
void rad_mpolar_f(
                    void          *A,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    float         *P,
                    rad_stride     ldP,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

P, real matrix, size m by n, output.

ldP, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := |\mathtt{A}[j * \mathtt{ldA} + k]|$ and $\mathtt{P}[j * \mathtt{ldP} + k] := \arg(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_mpolar_split_f

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.

Prototype

```
void rad_mpolar_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R,
                    rad_stride     ldR,
                    float         *P,
                    rad_stride     ldP,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

P, real matrix, size m by n, output.

ldP, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j*\mathtt{ldR}+k] := |\mathtt{A}[j*\mathtt{ldA}+k]|$ and $\mathtt{P}[j*\mathtt{ldP}+k] := \arg(\mathtt{A}[j*\mathtt{ldA}+k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vreal_f

Extract the real part of a complex vector.

Prototype

```
void rad_vreal_f(
                    void          *A,
                    rad_stride     strideA,
                    float         *R,
                    rad_stride     strideR,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ real(A$[j * \texttt{strideA}]$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_vreal_split_f

Extract the real part of a complex vector.

Prototype

```
void rad_vreal_split_f(
                   float         *A_re,
                   float         *A_im,
                   rad_stride     strideA,
                   float         *R,
                   rad_stride     strideR,
                   unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

R, real vector, length n, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R$[j * \texttt{strideR}] :=$ real(A$[j * \texttt{strideA}]$) where $0 \leqslant j < n$.

Restrictions

Errors

Notes

rad_mreal_f

Extract the real part of a complex matrix.

Prototype

```
void rad_mreal_f(
                   void          *A,
                   rad_stride     ldA,
                   float         *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{R}[j * \mathtt{ldR} + k] := \text{real}(\mathtt{A}[j * \mathtt{ldA} + k])$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

rad_vrect_f

Convert a pair of real vectors from complex polar to complex rectangular form.

Prototype

```
void rad_vrect_f(
                    float         *R,
                    rad_stride     strideR,
                    float         *P,
                    rad_stride     strideP,
                    void          *A,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

R, real vector, length n, input.

strideR, integer scalar, input.

P, real vector, length n, input.

strideP, integer scalar, input.

A, complex vector, length n, output.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{R}[j * \mathtt{strideR}] \cdot (\cos(\mathtt{P}[j * \mathtt{strideP}]) + \mathrm{i} \cdot \sin(\mathtt{P}[j * \mathtt{strideP}]))$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real vectors.

rad_vrect_split_f

Convert a pair of real vectors from complex polar to complex rectangular form.

Prototype

```
void rad_vrect_split_f(
                    float         *R,
                    rad_stride     strideR,
                    float         *P,
                    rad_stride     strideP,
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    unsigned int   n);
```

Parameters

R, real vector, length n, input.

strideR, integer scalar, input.

P, real vector, length n, input.

strideP, integer scalar, input.

A_re, real part of complex vector, length n, output.

A_im, imaginary part of complex vector, length n, output.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{R}[j * \mathtt{strideR}] \cdot (\cos(\mathtt{P}[j * \mathtt{strideP}]) + \mathrm{i} \cdot \sin(\mathtt{P}[j * \mathtt{strideP}]))$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real vectors.

rad_mrect_f

Convert a pair of real matrices from complex polar to complex rectangular form.

Prototype

```
void rad_mrect_f(
                    float         *R,
                    rad_stride     ldR,
                    float         *P,
                    rad_stride     ldP,
                    void          *A,
                    rad_stride     ldA,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

R, real matrix, size m by n, input.

ldR, integer scalar, input.

P, real matrix, size m by n, input.

ldP, integer scalar, input.

A, complex matrix, size m by n, output.

ldA, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{A}[j * \mathtt{ldA} + k] := \mathtt{R}[j * \mathtt{ldR} + k] \cdot (\cos(\mathtt{P}[j * \mathtt{ldP} + k]) + \mathrm{i} \cdot \sin(\mathtt{P}[j * \mathtt{ldP} + k]))$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real matrices.

rad_mrect_split_f

Convert a pair of real matrices from complex polar to complex rectangular form.

Prototype

```
void rad_mrect_split_f(
                    float         *R,
                    rad_stride     ldR,
                    float         *P,
                    rad_stride     ldP,
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

R, real matrix, size m by n, input.

ldR, integer scalar, input.

P, real matrix, size m by n, input.

ldP, integer scalar, input.

A_re, real part of complex matrix, size m by n, output.

A_im, imaginary part of complex matrix, size m by n, output.

ldA, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\texttt{A}[j * \texttt{ldA} + k] := \texttt{R}[j * \texttt{ldR} + k] \cdot (\cos(\texttt{P}[j * \texttt{ldP} + k]) + \mathrm{i} \cdot \sin(\texttt{P}[j * \texttt{ldP} + k]))$ where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

Errors

Notes

Complex numbers are always in rectangular format. The polar form is represented by two real matrices.

rad_Dvscatter_P

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.

Prototype

```
void rad_Dvscatter_P(
                  rad_scalar_P/void  *X,
                  rad_stride          strideX,
                  rad_scalar_P/void  *Y,
                  rad_stride          strideY,
                  unsigned int       *I,
                  rad_stride          strideI,
                  unsigned int        n);
```

The following instances are supported:

`rad_vscatter_f`

`rad_vscatter_i`

`rad_vscatter_si`

`rad_cvscatter_f`

Parameters

`X`, real or complex vector, length n, input.

`strideX`, integer scalar, input.

`Y`, real or complex vector, length m, output.

`strideY`, integer scalar, input.

`I`, vector-index vector, length n, input.

`strideI`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`Y`$[$`I`$[j]] :=$ `X`$[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

rad_cvscatter_split_f

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.

Prototype

```
void rad_cvscatter_split_f(
                    float          *X_re,
                    float          *X_im,
                    rad_stride      strideX,
                    float          *Y_re,
                    float          *Y_im,
                    rad_stride      strideY,
                    unsigned int   *I_re,
                    unsigned int   *I_im,
                    rad_stride      strideI,
                    unsigned int    n);
```

Parameters

`X_re`, real part of complex vector, length n, input.

`X_im`, imaginary part of complex vector, length n, input.

`strideX`, integer scalar, input.

`Y_re`, real part of complex vector, length m, output.

`Y_im`, imaginary part of complex vector, length m, output.

`strideY`, integer scalar, input.

`I_re`, real part of vector-index vector, length n, input.

`I_im`, imaginary part of vector-index vector, length n, input.

`strideI`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{Y}[\texttt{I}[j]] := \texttt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

rad_Dmscatter_P

The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.

Prototype

```
void rad_Dmscatter_P(
                  rad_scalar_P/void  *X,
                  rad_stride          strideX,
                  rad_scalar_P/void  *Y,
                  rad_stride          ldY,
                  rad_scalar_mi      *I,
                  rad_stride          strideI,
                  unsigned int        n);
```

The following instances are supported:

`rad_mscatter_f`

`rad_mscatter_i`

`rad_mscatter_si`

`rad_cmscatter_f`

Parameters

`X`, real or complex vector, length p, input.

`strideX`, integer scalar, input.

`Y`, real or complex matrix, size m by n, output.

`ldY`, integer scalar, input.

`I`, matrix-index vector, length p, input.

`strideI`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{Y}[\texttt{I}[j]] := \texttt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

rad_cmscatter_split_f

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.

Prototype

```
void rad_cmscatter_split_f(
                    float           *X_re,
                    float           *X_im,
                    rad_stride       strideX,
                    float           *Y_re,
                    float           *Y_im,
                    rad_stride       ldY,
                    rad_scalar_mi   *I_re,
                    rad_scalar_mi   *I_im,
                    rad_stride       strideI,
                    unsigned int     n);
```

Parameters

X_re, real part of complex vector, length p, input.

X_im, imaginary part of complex vector, length p, input.

strideX, integer scalar, input.

Y_re, real part of complex matrix, size m by n, output.

Y_im, imaginary part of complex matrix, size m by n, output.

ldY, integer scalar, input.

I_re, real part of matrix-index vector, length p, input.

I_im, imaginary part of matrix-index vector, length p, input.

strideI, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{Y}[\mathtt{I}[j]] := \mathtt{X}[j]$ where $0 \leqslant j < n$.

Restrictions

If the index vector contains duplicate entries, the value stored in the destination will be from the source vector, but which one is undefined. There is no in-place functionality for this function.

Errors

The arguments must conform to the following:

1. Index values in the index vector must be valid indexes into the output.

Notes

Values in the destination not indexed are not modified.

rad_Dvswap_P

Swap elements between two vectors.

Prototype

```
void rad_Dvswap_P (
                   rad_scalar_P/void  *A,
                   rad_stride          strideA,
                   rad_scalar_P/void  *B,
                   rad_stride          strideB,
                   unsigned int        n);
```

The following instances are supported:

rad_vswap_f

rad_cvswap_f

rad_vswap_i

rad_vswap_si

Parameters

A, real or complex vector, length n, modified in place.

strideA, integer scalar, input.

B, real or complex vector, length n, modified in place.

strideB, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$t := \mathtt{A}[j * \mathtt{strideA}]$
$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{B}[j * \mathtt{strideB}]$
$\mathtt{B}[j * \mathtt{strideB}] := t$
where $0 \leqslant j < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

rad_cvswap_split_f

Swap elements between two vectors.

Prototype

```
void rad_cvswap_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, modified in place.

A_im, imaginary part of complex vector, length n, modified in place.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, modified in place.

B_im, imaginary part of complex vector, length n, modified in place.

strideB, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$t := \mathtt{A}[j * \mathtt{strideA}]$
$\mathtt{A}[j * \mathtt{strideA}] := \mathtt{B}[j * \mathtt{strideB}]$
$\mathtt{B}[j * \mathtt{strideB}] := t$
where $0 \leqslant j < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

rad_Dmswap_P

Swap elements between two matrices.

Prototype

```
void rad_Dmswap_P (
                    rad_scalar_P/void  *A,
                    rad_stride          ldA,
                    rad_scalar_P/void  *B,
                    rad_stride          ldB,
                    unsigned int        m,
                    unsigned int        n);
```

The following instances are supported:

`rad_mswap_f`

`rad_mswap_i`

`rad_mswap_si`

`rad_cmswap_f`

Parameters

`A`, real or complex matrix, size m by n, modified in place.

`ldA`, integer scalar, input.

`B`, real or complex matrix, size m by n, modified in place.

`ldB`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$t := \texttt{A}[j * \texttt{ldA} + k]$
$\texttt{A}[j * \texttt{ldA} + k] := \texttt{B}[j * \texttt{ldB} + k]$
$\texttt{B}[j * \texttt{ldB} + k] := t$
where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

rad_cmswap_split_f

Swap elements between two matrices.

Prototype

```
void rad_cmswap_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     ldB,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, modified in place.

A_im, imaginary part of complex matrix, size m by n, modified in place.

ldA, integer scalar, input.

B_re, real part of complex matrix, size m by n, modified in place.

B_im, imaginary part of complex matrix, size m by n, modified in place.

ldB, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$t := \mathtt{A}[j * \mathtt{ldA} + k]$
$\mathtt{A}[j * \mathtt{ldA} + k] := \mathtt{B}[j * \mathtt{ldB} + k]$
$\mathtt{B}[j * \mathtt{ldB} + k] := t$
where $0 \leqslant j < m$ and $0 \leqslant k < n$.

Restrictions

This function may not be done in-place.

Errors

Notes

7 • Signal Processing Functions

7.1 FFT Functions

rad_rcfftmop_f

rad_rcfftmop_split_f

rad_ccfft2dop_create_f

rad_ccfft2dip_create_f

rad_crfft2dop_create_f

rad_rcfft2dop_create_f

rad_ccfft2dip_f

rad_ccfft2dip_split_f

rad_ccfft2dop_f

rad_ccfft2dop_split_f

rad_crfft2dop_f

rad_crfft2dop_split_f

rad_rcfft2dop_f

rad_rcfft2dop_split_f

rad_fft2d_destroy_f

rad_fft2d_getattr_f

rad_ccfftip_create_f

Create a 1D FFT object.

Prototype

```
rad_fft_f * rad_ccfftip_create_f(
                  unsigned int  length,
                  float         scale,
                  rad_fft_dir   dir,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`length`, vector-index scalar, input. The length N of the data vector.

`scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.

`dir`, enumerated type, input.

`RAD_FFT_FWD`	forward
`RAD_FFT_INV`	reverse (or inverse)

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: complex-to-complex in-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.
`NULL` is returned if the create fails.

Restrictions
Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive and non-zero.
2. `dir` must be valid.
3. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

rad_ccfftop_create_f

Create a 1D FFT object.

Prototype

```
rad_fft_f * rad_ccfftop_create_f(
                  unsigned int  length,
                  float         scale,
                  rad_fft_dir   dir,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`length`, vector-index scalar, input. The length N of the data vector.

`scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.

`dir`, enumerated type, input.

`RAD_FFT_FWD`	forward
`RAD_FFT_INV`	reverse (or inverse)

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: complex-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.
`NULL` is returned if the create fails.

Restrictions
Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive and non-zero.
2. `dir` must be valid.
3. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

rad_crfftop_create_f

Create a 1D FFT object.

Prototype

```
rad_fft_f * rad_crfftop_create_f(
                  unsigned int  length,
                  float         scale,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`length`, vector-index scalar, input. The length N of the data vector.

`scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: (reverse) complex-to-real out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.
`NULL` is returned if the create fails.

Restrictions

The length N must be even.

Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

rad_rcfftop_create_f

Create a 1D FFT object.

Prototype

```
rad_fft_f * rad_rcfftop_create_f(
                  unsigned int  length,
                  float         scale,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`length`, vector-index scalar, input. The length N of the data vector.

`scale`, real scalar, input. Typical scale factors are 1, $1/N$ and $1/\sqrt{N}$.

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D FFT object holding the information on the type of FFT to be computed: (forward) real-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$.
`NULL` is returned if the create fails.

Restrictions

The length N must be even.

Errors

The arguments must conform to the following:

1. `length`, the length of the FFT, must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported on vectors of any length.

rad_ccfftip_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftip_f(
                   rad_fft_f   *plan,
                   void        *xy,
                   rad_stride   stridexy);
```

Parameters

`plan`, structure, input.

`xy`, complex vector, modified in place.

`stridexy`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex in-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place FFT object.
3. The input must be a complex vector of length N, conformant to the FFT object.

Notes

rad_ccfftip_split_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftip_split_f(
                    rad_fft_f   *plan,
                    float       *xy_re,
                    float       *xy_im,
                    rad_stride   stridexy);
```

Parameters

`plan`, structure, input.

`xy_re`, real part of complex vector, modified in place.

`xy_im`, imaginary part of complex vector, modified in place.

`stridexy`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex in-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place FFT object.
3. The input must be a complex vector of length N, conformant to the FFT object.

Notes

rad_ccfftop_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftop_f(
                   rad_fft_f   *plan,
                   void        *x,
                   rad_stride   stridex,
                   void        *y,
                   rad_stride   stridey);
```

Parameters

`plan`, structure, input.

`x`, complex vector, input.

`stridex`, integer scalar, input.

`y`, complex vector, output.

`stridey`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex out-of-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place FFT object.
3. The input and output must be complex vectors of length N, conformant to the FFT object.
4. The input and output vectors must not overlap.

Notes

rad_ccfftop_split_f

Apply a complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftop_split_f(
                    rad_fft_f   *plan,
                    float       *x_re,
                    float       *x_im,
                    rad_stride   stridex,
                    float       *y_re,
                    float       *y_im,
                    rad_stride   stridey);
```

Parameters

`plan`, structure, input.

`x_re`, real part of complex vector, input.

`x_im`, imaginary part of complex vector, input.

`stridex`, integer scalar, input.

`y_re`, real part of complex vector, output.

`y_im`, imaginary part of complex vector, output.

`stridey`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex out-of-place Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the complex vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$
where $W_N = \exp(sign \cdot 2\pi \mathrm{i}/N)$ and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place FFT object.
3. The input and output must be complex vectors of length N, conformant to the FFT object.
4. The input and output vectors must not overlap.

Notes

rad_crfftop_f

Apply a complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void rad_crfftop_f(
                   rad_fft_f   *plan,
                   void        *x,
                   rad_stride   stridex,
                   float       *y,
                   rad_stride   stridey);
```

Parameters

`plan`, structure, input.

`x`, complex vector, input.

`stridex`, integer scalar, input.

`y`, real vector, output.

`stridey`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-real out-of-place (reverse) Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the real vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(+2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride vectors are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place FFT object.
3. The input must be a complex vector of length $N/2 + 1$, conformant to the FFT object.
4. The output must be a real vector of even length N, conformant to the FFT object.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work.
For the output sequence, y, to be a real sequence, the following identity on the input sequence, x, must be true: $x_j = x^*_{N-j}$ for $\lfloor N/2 \rfloor < j < N$.

The input values x_j for $j > \lfloor N/2 \rfloor$ need not be supplied; they can be inferred from the first half of the input.

Thus, in the complex-to-real routine, x is a complex vector of length $\lfloor N/2 \rfloor + 1$ and y is a real vector of length N. Even though only $\lfloor N/2 \rfloor + 1$ input complex values are supplied, the size of the transform is still N in this case, because implicitly you are using the FFT formula for a sequence of length N.

The first value of the input vector, $x[0]$ must be a real number (that is, it must have zero imaginary part). The first value corresponds to the zero (DC) frequency component of the data. Since we restrict N to be an even number, the last value of the input vector, $x[N/2]$, must also be real. The last value corresponds to one half the Nyquist rate (or sample rate). This value is sometimes called the folding frequency. The routine assumes that these values are real; if you specify a non-zero imaginary part, it is ignored.

rad_crfftop_split_f

Apply a complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void rad_crfftop_split_f(
                    rad_fft_f   *plan,
                    float       *x_re,
                    float       *x_im,
                    rad_stride   stridex,
                    float       *y,
                    rad_stride   stridey);
```

Parameters

`plan`, structure, input.

`x_re`, real part of complex vector, input.

`x_im`, imaginary part of complex vector, input.

`stridex`, integer scalar, input.

`y`, real vector, output.

`stridey`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-real out-of-place (reverse) Fast Fourier Transform (FFT) of the complex vector x, and stores the results in the real vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(+2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride vectors are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place FFT object.
3. The input must be a complex vector of length $N/2 + 1$, conformant to the FFT object.
4. The output must be a real vector of even length N, conformant to the FFT object.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work.

For the output sequence, y, to be a real sequence, the following identity on the input sequence, x, must be true: $x_j = x^*_{N-j}$ for $\lfloor N/2 \rfloor < j < N$.
The input values x_j for $j > \lfloor N/2 \rfloor$ need not be supplied; they can be inferred from the first half of the input.
Thus, in the complex-to-real routine, x is a complex vector of length $\lfloor N/2 \rfloor + 1$ and y is a real vector of length N. Even though only $\lfloor N/2 \rfloor + 1$ input complex values are supplied, the size of the transform is still N in this case, because implicitly you are using the FFT formula for a sequence of length N.
The first value of the input vector, $x[0]$ must be a real number (that is, it must have zero imaginary part). The first value corresponds to the zero (DC) frequency component of the data. Since we restrict N to be an even number, the last value of the input vector, $x[N/2]$, must also be real. The last value corresponds to one half the Nyquist rate (or sample rate). This value is sometimes called the folding frequency. The routine assumes that these values are real; if you specify a non-zero imaginary part, it is ignored.

rad_rcfftop_f

Apply a real-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_rcfftop_f(
                   rad_fft_f   *plan,
                   float       *x,
                   rad_stride   stridex,
                   void        *y,
                   rad_stride   stridey);
```

Parameters

`plan`, structure, input.

`x`, real vector, input.

`stridex`, integer scalar, input.

`y`, complex vector, output.

`stridey`, integer scalar, input.

Return Value

none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real vector x, and stores the results in the complex vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(-2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride views are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place FFT object.
3. The input must be a real vector of even length N.
4. The output must be a complex vector of length $N/2 + 1$.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.
The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. The theory of Fourier transforms tells us that for real input data, you have to compute only the first $\lfloor N/2 \rfloor + 1$

complex output values because the remaining values can be computed from the first half by the simple formula: $y_k = y^*_{N-k}$ for $\lfloor N/2 \rfloor < k < N$.

For real input data, the first output value, $y[0]$, will always be a real number (the imaginary part will be zero). The first output value is sometimes called the DC component of the FFT and corresponds to zero frequency. Since we restrict N to be an even number, $y[N/2]$ will also be real and thus, have zero imaginary part. The last value is called the folding frequency and is equal to one half the sample rate of the input data.

Thus, in the real-to-complex routine, x is a real array of even length N and y is a complex array of length $N/2 + 1$.

rad_rcfftop_split_f

Apply a real-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_rcfftop_split_f(
                    rad_fft_f   *plan,
                    float       *x,
                    rad_stride   stridex,
                    float       *y_re,
                    float       *y_im,
                    rad_stride   stridey);
```

Parameters

`plan`, structure, input.

`x`, real vector, input.

`stridex`, integer scalar, input.

`y_re`, real part of complex vector, output.

`y_im`, imaginary part of complex vector, output.

`stridey`, integer scalar, input.

Return Value

none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real vector x, and stores the results in the complex vector y.
$y[k] := scale \cdot \sum_{j=0}^{N-1} x[j] \cdot (W_N)^{kj}$ where $W_N = \exp(-2\pi \mathrm{i}/N)$.

Restrictions

Only unit stride views are supported. The length, N, must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place FFT object.
3. The input must be a real vector of even length N.
4. The output must be a complex vector of length $N/2 + 1$.
5. The input and output vectors must not overlap.
6. The input and output vectors must be unit stride.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.
The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. The theory of

Fourier transforms tells us that for real input data, you have to compute only the first $\lfloor N/2 \rfloor + 1$ complex output values because the remaining values can be computed from the first half by the simple formula: $y_k = y^*_{N-k}$ for $\lfloor N/2 \rfloor < k < N$.

For real input data, the first output value, $y[0]$, will always be a real number (the imaginary part will be zero). The first output value is sometimes called the DC component of the FFT and corresponds to zero frequency. Since we restrict N to be an even number, $y[N/2]$ will also be real and thus, have zero imaginary part. The last value is called the folding frequency and is equal to one half the sample rate of the input data.

Thus, in the real-to-complex routine, x is a real array of even length N and y is a complex array of length $N/2 + 1$.

rad_fft_destroy_f

Destroy an FFT object.

Prototype

```
int rad_fft_destroy_f(
                    rad_fft_f *plan);
```

Parameters

`plan`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) an FFT object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The input object must conform to the following:

1. The FFT object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_fft_getattr_f

Return the attributes of an FFT object.

Prototype

```
void rad_fft_getattr_f(
                   rad_fft_f      *plan,
                   rad_fft_attr_f *attr);
```

Parameters

`plan`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`unsigned int`	`input`	input length
`unsigned int`	`output`	output length
`rad_fft_place`	`place`	in-place or out-of-place
`float`	`scale`	scale factor
`rad_fft_dir`	`dir`	forward or reverse

Return Value

none.

Description

Returns the attributes of an FFT object.

Restrictions
Errors

The arguments must conform to the following:

1. The FFT object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

There is no attribute that explicitly indicates complex-to-complex, real-to-complex, or complex-to-real FFTs. This may be inferred by examining the input and output sizes.

rad_ccfftmop_create_f

Create a 1D multiple FFT object.

Prototype

```
rad_fftm_f * rad_ccfftmop_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  rad_fft_dir   dir,
                  rad_major     major,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).

`cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).

`scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.

`dir`, enumerated type, input.

`RAD_FFT_FWD`	forward
`RAD_FFT_INV`	reverse (or inverse)

`major`, enumerated type, input.

`RAD_ROW`	apply operation to the rows
`RAD_COL`	apply operation to the columns

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: complex-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.
`NULL` is returned if the create fails.

Restrictions
Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `dir` must be valid.

3. `major` must be valid.

4. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.
FFT operations are supported for all values of N and M.

rad_ccfftmip_create_f

Create a 1D multiple FFT object.

Prototype

```
rad_fftm_f * rad_ccfftmip_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  rad_fft_dir   dir,
                  rad_major     major,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).

`cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).

`scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.

`dir`, enumerated type, input.

`RAD_FFT_FWD`	forward
`RAD_FFT_INV`	reverse (or inverse)

`major`, enumerated type, input.

`RAD_ROW`	apply operation to the rows
`RAD_COL`	apply operation to the columns

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: complex-to-complex in-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.
`NULL` is returned if the create fails.

Restrictions
Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `dir` must be valid.

3. `major` must be valid.

4. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.
FFT operations are supported for all values of N and M.

rad_crfftmop_create_f

Create a 1D multiple FFT object.

Prototype

```
rad_fftm_f * rad_crfftmop_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  rad_major     major,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).

`cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).

`scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.

`major`, enumerated type, input.

`RAD_ROW`	apply operation to the rows
`RAD_COL`	apply operation to the columns

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: (reverse) complex-to-real out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.
`NULL` is returned if the create fails.

Restrictions

In the FFT direction, the length must be even and the stride must be 1.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `major` must be valid.
3. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.

FFT operations are supported for all values of N and M.

rad_rcfftmop_create_f

Create a 1D multiple FFT object.

Prototype

```
rad_fftm_f * rad_rcfftmop_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  rad_major     major,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`rows`, vector-index scalar, input. Length of column FFT or number of row FFT's (M).

`cols`, vector-index scalar, input. Length of row FFT or number of column FFT's (N).

`scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/\sqrt{M}$ and $1/\sqrt{N}$.

`major`, enumerated type, input.

`RAD_ROW`	apply operation to the rows
`RAD_COL`	apply operation to the columns

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 1D multiple FFT object holding the information on the type of FFT to be computed: (forward) real-to-complex out-of-place. The 1D FFT object is used to compute a Fast Fourier Transform (FFT) of a vector x, and store the results in a vector y.
A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors.
`NULL` is returned if the create fails.

Restrictions

In the FFT direction, the length must be even and the stride must be 1.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive.
2. `major` must be valid.
3. `hint` must be valid.

Notes

Performing a 1D FFT on the data by rows and then by columns (or vice versa) is equivalent to performing a 2D FFT on the matrix. This would require two multiple FFT objects, one by rows and one by columns.

FFT operations are supported for all values of N and M.

rad_fftm_destroy_f

Destroy an FFT object.

Prototype

```
int rad_fftm_destroy_f(
                    rad_fftm_f *plan);
```

Parameters

`plan`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) an FFT object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The input object must conform to the following:

1. The FFT object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_fftm_getattr_f

Return the attributes of an FFT object.

Prototype

```
void rad_fftm_getattr_f(
                    rad_fftm_f      *plan,
                    rad_fftm_attr_f *attr);
```

Parameters

`plan`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`rad_scalar_mi`	`input`	numbers of rows and columns in input matrix
`rad_scalar_mi`	`output`	numbers of rows and columns in output matrix
`rad_fft_place`	`place`	in-place or out-of-place
`float`	`scale`	scale factor
`rad_fft_dir`	`dir`	forward or reverse
`rad_major`	`major`	by row or column

Return Value

none.

Description

Returns the attributes of an FFT object.

Restrictions
Errors

The arguments must conform to the following:

1. The FFT object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

There is no attribute that explicitly indicates complex-to-complex, real-to-complex, or complex-to-real FFTs. This may be inferred by examining the input and output sizes.

rad_ccfftmip_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftmip_f(
                    rad_fftm_f *plan,
                    void       *XY,
                    rad_stride  ldXY);
```

Parameters

`plan`, structure, input.

`XY`, complex matrix, modified in place.

`ldXY`, integer scalar, input.

Return Value

none.

Description

Computes multiple complex-to-complex in-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.
A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.
See `rad_ccfftip_split_f` for more details.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place multiple FFT object.
3. The input must be a complex matrix of size M by N, conformant to the FFT object.

Notes

rad_ccfftmip_split_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftmip_split_f(
                    rad_fftm_f *plan,
                    float      *XY_re,
                    float      *XY_im,
                    rad_stride  ldXY);
```

Parameters

`plan`, structure, input.

`XY_re`, real part of complex matrix, modified in place.

`XY_im`, imaginary part of complex matrix, modified in place.

`ldXY`, integer scalar, input.

Return Value

none.

Description

Computes multiple complex-to-complex in-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.
A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.
See `rad_ccfftip_split_f` for more details.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place multiple FFT object.
3. The input must be a complex matrix of size M by N, conformant to the FFT object.

Notes

rad_ccfftmop_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftmop_f(
                    rad_fftm_f *plan,
                    void       *X,
                    rad_stride  ldX,
                    void       *Y,
                    rad_stride  ldY);
```

Parameters

`plan`, structure, input.

`X`, complex matrix, input.

`ldX`, integer scalar, input.

`Y`, complex matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes multiple complex-to-complex out-of-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.
A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.
See `rad_ccfftop_split_f` for more details.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place multiple FFT object.
3. The input and output must be complex matrices of size M by N, conformant to the FFT object.
4. The input and output matrices must not overlap.

Notes

rad_ccfftmop_split_f

Apply a multiple complex-to-complex Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfftmop_split_f(
                    rad_fftm_f *plan,
                    float      *X_re,
                    float      *X_im,
                    rad_stride  ldX,
                    float      *Y_re,
                    float      *Y_im,
                    rad_stride  ldY);
```

Parameters

plan, structure, input.

X_re, real part of complex matrix, input.

X_im, imaginary part of complex matrix, input.

ldX, integer scalar, input.

Y_re, real part of complex matrix, output.

Y_im, imaginary part of complex matrix, output.

ldY, integer scalar, input.

Return Value

none.

Description

Computes multiple complex-to-complex out-of-place Fast Fourier Transforms (FFTs) of the complex vectors in matrix X, and stores the results in the complex matrix Y.
A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors. The major direction (row or column) is specified in the creation of the FFT object.
See rad_ccfftop_split_f for more details.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place multiple FFT object.
3. The input and output must be complex matrices of size M by N, conformant to the FFT object.
4. The input and output matrices must not overlap.

Notes

rad_crfftmop_f

Apply a multiple complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void rad_crfftmop_f(
                    rad_fftm_f *plan,
                    void       *X,
                    rad_stride  ldX,
                    float      *Y,
                    rad_stride  ldY);
```

Parameters

`plan`, structure, input.

`X`, complex matrix, input.

`ldX`, integer scalar, input.

`Y`, real matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-real out-of-place reverse Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.
A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.
See `rad_crfftop_f` for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The output length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place multiple FFT object.
3. The input must be a complex matrix of size:

   By rows: M by $N/2+1$, N even.
   By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The output must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

rad_crfftmop_split_f

Apply a multiple complex-to-real Fast Fourier Transform (FFT).

Prototype

```
void rad_crfftmop_split_f(
                    rad_fftm_f *plan,
                    float      *X_re,
                    float      *X_im,
                    rad_stride  ldX,
                    float      *Y,
                    rad_stride  ldY);
```

Parameters

`plan`, structure, input.

`X_re`, real part of complex matrix, input.

`X_im`, imaginary part of complex matrix, input.

`ldX`, integer scalar, input.

`Y`, real matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-real out-of-place reverse Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.
A series of 1D complex vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.
See `rad_crfftop_f` for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The output length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place multiple FFT object.
3. The input must be a complex matrix of size:

   By rows: M by $N/2+1$, N even.
   By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The output must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

rad_rcfftmop_f

Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT).

Prototype

```
void rad_rcfftmop_f(
                    rad_fftm_f *plan,
                    float      *X,
                    rad_stride  ldX,
                    void       *Y,
                    rad_stride  ldY);
```

Parameters

`plan`, structure, input.

`X`, real matrix, input.

`ldX`, integer scalar, input.

`Y`, complex matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.
A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.
See `rad_rcfftop_f` for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The input length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place multiple FFT object.
3. The output must be a complex matrix of size:

   By rows: M by $N/2+1$, N even.
   By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The input must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

rad_rcfftmop_split_f

Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT).

Prototype

```
void rad_rcfftmop_split_f(
                    rad_fftm_f *plan,
                    float      *X,
                    rad_stride  ldX,
                    float      *Y_re,
                    float      *Y_im,
                    rad_stride  ldY);
```

Parameters

`plan`, structure, input.

`X`, real matrix, input.

`ldX`, integer scalar, input.

`Y_re`, real part of complex matrix, output.

`Y_im`, imaginary part of complex matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a real-to-complex out-of-place (forward) Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.
A series of 1D real vectors is stored in a matrix object in row major or column major order. Multiple 1D FFTs are then performed on the series of vectors (which must have unit stride). The major direction (row or column) is specified in the creation of the FFT object.
See `rad_rcfftop_f` for more details.

Restrictions

Only unit stride along the specified row or column FFT direction is supported. The input length of the individual FFTs must be even.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place multiple FFT object.
3. The output must be a complex matrix of size:

   By rows: M by $N/2+1$, N even.
   By columns: $M/2+1$ by N, M even.

   M by N are obtained from the FFT object.
4. The input must be a real matrix of size M by N, conformant to the FFT object.
5. The input and output matrices must not overlap.
6. The input and output matrices must be unit-stride in the transform direction.

Notes

rad_ccfft2dop_create_f

Create a 2D FFT object.

Prototype

```
rad_fft2d_f * rad_ccfft2dop_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  rad_fft_dir   dir,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

rows, vector-index scalar, input. Number of rows (M).

cols, vector-index scalar, input. Number of columns (N).

scale, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.

dir, enumerated type, input.

RAD_FFT_FWD	forward
RAD_FFT_INV	reverse (or inverse)

ntimes, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

hint, enumerated type, input.

RAD_ALG_SPACE	minimise memory usage
RAD_ALG_TIME	minimise execution time
RAD_ALG_NOISE	maximise numerical accuracy

Return Value

structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: complex-to-complex out-of-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.
NULL is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. rows and cols must be positive and non-zero.
2. dir must be valid.
3. hint must be valid.

Notes

FFT operations are supported for all legal values of M and N.

rad_ccfft2dip_create_f

Create a 2D FFT object.

Prototype

```
rad_fft2d_f * rad_ccfft2dip_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  rad_fft_dir   dir,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

rows, vector-index scalar, input. Number of rows (M).

cols, vector-index scalar, input. Number of columns (N).

scale, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.

dir, enumerated type, input.

RAD_FFT_FWD	forward
RAD_FFT_INV	reverse (or inverse)

ntimes, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

hint, enumerated type, input.

RAD_ALG_SPACE	minimise memory usage
RAD_ALG_TIME	minimise execution time
RAD_ALG_NOISE	maximise numerical accuracy

Return Value

structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: complex-to-complex in-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.
NULL is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. rows and cols must be positive and non-zero.
2. dir must be valid.
3. hint must be valid.

Notes

FFT operations are supported for all legal values of M and N.

rad_crfft2dop_create_f

Create a 2D FFT object.

Prototype

```
rad_fft2d_f * rad_crfft2dop_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`rows`, vector-index scalar, input. Number of rows (M).

`cols`, vector-index scalar, input. Number of columns (N).

`scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: (reverse) complex-to-real out-of-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.
`NULL` is returned if the create fails.

Restrictions

The FFT is restricted to views with unit stride in either the row or column direction. The lengths M and N must be even.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported for all legal values of M and N.

rad_rcfft2dop_create_f

Create a 2D FFT object.

Prototype

```
rad_fft2d_f * rad_rcfft2dop_create_f(
                  unsigned int  rows,
                  unsigned int  cols,
                  float         scale,
                  unsigned int  ntimes,
                  rad_alg_hint  hint);
```

Parameters

`rows`, vector-index scalar, input. Number of rows (M).

`cols`, vector-index scalar, input. Number of columns (N).

`scale`, real scalar, input. Typical scale factors are 1, $1/M$, $1/N$, $1/(MN)$, $1/\sqrt{N}$, $1/\sqrt{N}$ and $1/\sqrt{MN}$.

`ntimes`, integer scalar, input. An estimate of how many times the FFT object will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

Return Value

structure.

Description

Creates a 2D FFT object holding the information on the type of FFT to be computed: (forward) real-to-complex out-of-place. The 2D FFT object is used to compute Fast Fourier Transforms (FFTs) of a matrix X, and stores the results in a matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N.
`NULL` is returned if the create fails.

Restrictions

The FFT is restricted to views with unit stride in either the row or column direction. The lengths M and N must be even.

Errors

The arguments must conform to the following:

1. `rows` and `cols` must be positive, even and non-zero.
2. `hint` must be valid.

Notes

FFT operations are supported for all legal values of M and N.

rad_ccfft2dip_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfft2dip_f(
                    rad_fft2d_f *plan,
                    void        *XY,
                    rad_stride   ldXY);
```

Parameters

`plan`, structure, input.

`XY`, complex matrix, modified in place.

`ldXY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex in-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place 2D FFT object.
3. The input/output `XY` must be a complex matrix of size M by N (conformant with the 2D FFT object).

Notes

rad_ccfft2dip_split_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfft2dip_split_f(
                    rad_fft2d_f *plan,
                    float       *XY_re,
                    float       *XY_im,
                    rad_stride   ldXY);
```

Parameters

`plan`, structure, input.

`XY_re`, real part of complex matrix, modified in place.

`XY_im`, imaginary part of complex matrix, modified in place.

`ldXY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex in-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex in-place 2D FFT object.
3. The input/output `XY` must be a complex matrix of size M by N (conformant with the 2D FFT object).

Notes

rad_ccfft2dop_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfft2dop_f(
                    rad_fft2d_f *plan,
                    void        *X,
                    rad_stride   ldX,
                    void        *Y,
                    rad_stride   ldY);
```

Parameters

`plan`, structure, input.

`X`, complex matrix, input.

`ldX`, integer scalar, input.

`Y`, complex matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex out-of-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place 2D FFT object.
3. The input `X` must be a complex matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.

Notes

rad_ccfft2dop_split_f

Apply a complex-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_ccfft2dop_split_f(
                    rad_fft2d_f *plan,
                    float       *X_re,
                    float       *X_im,
                    rad_stride   ldX,
                    float       *Y_re,
                    float       *Y_im,
                    rad_stride   ldY);
```

Parameters

`plan`, structure, input.

`X_re`, real part of complex matrix, input.

`X_im`, imaginary part of complex matrix, input.

`ldX`, integer scalar, input.

`Y_re`, real part of complex matrix, output.

`Y_im`, imaginary part of complex matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-complex out-of-place 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the complex matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(sign \cdot 2\pi \mathrm{i}/M)$; similarly for W_N and $sign$ is -1 for a forward transform and $+1$ for a reverse transform.

Restrictions
Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-complex out-of-place 2D FFT object.
3. The input `X` must be a complex matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.

Notes

rad_crfft2dop_f

Apply a complex-to-real 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_crfft2dop_f(
                    rad_fft2d_f *plan,
                    void        *X,
                    rad_stride   ldX,
                    float       *Y,
                    rad_stride   ldY);
```

Parameters

`plan`, structure, input.

`X`, complex matrix, input.

`ldX`, integer scalar, input.

`Y`, real matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-real out-of-place (reverse) 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(+2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix `Y` must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place 2D FFT object.
3. The input `X` must be a complex matrix of size (conformant with the 2D FFT object):

   By rows: M by $N/2+1$ where N is even
   By columns: $M/2+1$ by N where M is even.
4. The output `Y` must be a real matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.
6. The output matrix must be unit stride in the row or column direction.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work. See `rad_crfftop_f` for more details.

rad_crfft2dop_split_f

Apply a complex-to-real 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_crfft2dop_split_f(
                    rad_fft2d_f *plan,
                    float       *X_re,
                    float       *X_im,
                    rad_stride   ldX,
                    float       *Y,
                    rad_stride   ldY);
```

Parameters

`plan`, structure, input.

`X_re`, real part of complex matrix, input.

`X_im`, imaginary part of complex matrix, input.

`ldX`, integer scalar, input.

`Y`, real matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a complex-to-real out-of-place (reverse) 2D Fast Fourier Transform (FFT) of the complex matrix X, and stores the results in the real matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(+2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix `Y` must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a complex-to-real out-of-place 2D FFT object.
3. The input `X` must be a complex matrix of size (conformant with the 2D FFT object):

   By rows: M by $N/2+1$ where N is even

   By columns: $M/2+1$ by N where M is even.
4. The output `Y` must be a real matrix of size M by N (conformant with the 2D FFT object).
5. The input and output matrices must not overlap.
6. The output matrix must be unit stride in the row or column direction.

Notes

Generally, the FFT transforms a complex sequence into a complex sequence. However, in certain applications we may know the output sequence is real. Often, this is the case because the complex input sequence was the transform of a real sequence. In this case, you can save about half of the computational work. See `rad_crfftop_f` for more details.

rad_rcfft2dop_f

Apply a real-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_rcfft2dop_f(
                    rad_fft2d_f *plan,
                    float       *X,
                    rad_stride   ldX,
                    void        *Y,
                    rad_stride   ldY);
```

Parameters

`plan`, structure, input.

`X`, real matrix, input.

`ldX`, integer scalar, input.

`Y`, complex matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a real-to-complex out-of-place (forward) 2D Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(-2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix `X` must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place 2D FFT object.
3. The input `X` must be a real matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size (conformant with the 2D FFT object):

   By rows: M by $N/2+1$ where N is even

   By columns: $M/2+1$ by N where M is even.
5. The input and output matrices must not overlap.
6. The input matrix must be unit stride in the row or column direction.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.

This routine computes a real-to-complex transform along the unit stride dimension, followed by a complex-to-complex transform in the other dimension. The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. See `rad_rcfftop_f` for more details.

rad_rcfft2dop_split_f

Apply a real-to-complex 2D Fast Fourier Transform (FFT).

Prototype

```
void rad_rcfft2dop_split_f(
                    rad_fft2d_f *plan,
                    float       *X,
                    rad_stride   ldX,
                    float       *Y_re,
                    float       *Y_im,
                    rad_stride   ldY);
```

Parameters

`plan`, structure, input.

`X`, real matrix, input.

`ldX`, integer scalar, input.

`Y_re`, real part of complex matrix, output.

`Y_im`, imaginary part of complex matrix, output.

`ldY`, integer scalar, input.

Return Value

none.

Description

Computes a real-to-complex out-of-place (forward) 2D Fast Fourier Transform (FFT) of the real matrix X, and stores the results in the complex matrix Y.
$Y[j,k] := scale \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[m,n] \cdot (W_N)^{mj} (W_N)^{nk}$ where $W_M = \exp(-2\pi \mathrm{i}/M)$; similarly for W_N.

Restrictions

Matrix `X` must be unit stride in one direction, and it must have even length in this direction.

Errors

The arguments must conform to the following:

1. All objects must be valid.
2. The FFT object must be a real-to-complex out-of-place 2D FFT object.
3. The input `X` must be a real matrix of size M by N (conformant with the 2D FFT object).
4. The output `Y` must be a complex matrix of size (conformant with the 2D FFT object):

   By rows: M by $N/2+1$ where N is even

   By columns: $M/2+1$ by N where M is even.
5. The input and output matrices must not overlap.
6. The input matrix must be unit stride in the row or column direction.

Notes

The mathematical definition of the Fourier transform takes a sequence of N complex values and transforms it to another sequence of N complex values. A complex-to-complex FFT routine will take N complex inputs, and produce N complex outputs.

This routine computes a real-to-complex transform along the unit stride dimension, followed by a complex-to-complex transform in the other dimension. The purpose of a separate real-to-complex FFT routine is efficiency. Since the input data are real, you can make use of this fact to save almost half of the computational work. See `rad_rcfftop_f` for more details.

rad_fft2d_destroy_f

Destroy an FFT object.

Prototype

```
int rad_fft2d_destroy_f(
                    rad_fft2d_f *plan);
```

Parameters

`plan`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) an FFT object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The input object must conform to the following:

1. The FFT object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_fft2d_getattr_f

Return the attributes of an FFT object.

Prototype

```
void rad_fft2d_getattr_f(
                   rad_fft2d_f      *plan,
                   rad_fft2d_attr_f *attr);
```

Parameters

`plan`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`rad_scalar_mi`	`input`	input length
`rad_scalar_mi`	`output`	output length
`rad_fft_place`	`place`	in-place or out-of-place
`float`	`scale`	scale factor
`rad_fft_dir`	`dir`	forward or reverse

Return Value

none.

Description

Returns the attributes of an FFT object.

Restrictions
Errors

The arguments must conform to the following:

1. The FFT object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

There is no attribute that explicitly indicates complex-to-complex, real-to-complex, or complex-to-real FFTs. This may be inferred by examining the input and output sizes.

7.2 Convolution/Correlation Functions

rad_conv1d_create_f

Create a decimated 1D convolution filter object.

Prototype

```
rad_conv1d_f * rad_conv1d_create_f(
                   float              *kernel,
                   rad_stride          stridekernel,
                   rad_symmetry        symm,
                   unsigned int        N,
                   unsigned int        D,
                   rad_support_region  support,
                   unsigned int        ntimes,
                   rad_alg_hint        hint,
                   unsigned int        n);
```

Parameters

`kernel`, real vector, input. Vector of non-redundant filter coefficients. There are M in the non-symmetric case and $\lceil M/2 \rceil$ for symmetric filters.

`stridekernel`, integer scalar, input.

`symm`, enumerated type, input.

`RAD_NONSYM`	non-symmetric
`RAD_SYM_EVEN_LEN_ODD`	(even) symmetric, odd length
`RAD_SYM_EVEN_LEN_EVEN`	(even) symmetric, even length

`N`, integer scalar, input. Length of data vector.

`D`, integer scalar, input. Decimation factor.

`support`, enumerated type, input.

`RAD_SUPPORT_FULL`	maximum region
`RAD_SUPPORT_SAME`	input and output same size
`RAD_SUPPORT_MIN`	region without zero extending the kernel

`ntimes`, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

`n`, integer scalar, input.

Return Value

structure.

Description

Creates a decimated convolution filter object and returns a pointer to the object. The user specifies the kernel (filter order, symmetry, and filter coefficients), the region of support, and the integral output decimation factor.
If the create fails, `NULL` is returned.

A 1D convolution object is used to compute the convolution of a real filter (kernel) vector h of length M with a real data vector x of length N and output decimation factor D to produce an output vector y.

Full:
length $= \lfloor (N + M - 2)/D \rfloor + 1$

$$y[k] := \sum_{j=0}^{M-1} h[j] \, \langle x[kD - j] \rangle \qquad \text{for } 0 \leqslant k \leqslant \lfloor (N + M - 2)/D \rfloor.$$

Same size:
length $= \lfloor (N - 1)/D \rfloor + 1$

$$y[k] := \sum_{j=0}^{M-1} h[j] \langle x[kD + \lfloor M/2 \rfloor - j] \rangle \qquad \text{for } 0 \leqslant k \leqslant \lfloor (N - 1)/D \rfloor.$$

Minimum (not zero padded):
length $= \lfloor (N - 1)/D \rfloor - \lfloor (M - 1)/D \rfloor + 1$

$$y[k] := \sum_{j=0}^{M-1} h[j] \langle x[kD + (M - 1) - j] \rangle \qquad \text{for } 0 \leqslant k \leqslant \lfloor (N - 1)/D \rfloor - \lfloor (M - 1)/D \rfloor.$$

Where $\langle x[j] \rangle = \begin{cases} x[j] & : \quad 0 \leqslant j < N \\ 0 & : \quad \text{otherwise.} \end{cases}$

If the kernel is symmetric the redundant values are not specified.

Restrictions

The filter length must be less than or equal to the data length, $M \leqslant N$.

Errors

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments. The decimation factor, D, is normally one for non-lowpass filters.

rad_conv1d_destroy_f

Destroy a 1D convolution object.

Prototype

```
int rad_conv1d_destroy_f(
                    rad_conv1d_f *plan);
```

Parameters

`plan`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) a 1D convolution object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The arguments must conform to the following:

1. The 1D convolution object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_conv1d_getattr_f

Returns the attributes for a 1D convolution object.

Prototype

```
void rad_conv1d_getattr_f(
                  rad_conv1d_f      *plan,
                  rad_conv1d_attr_f *attr);
```

Parameters

`plan`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`unsigned int`	`kernel_len`	kernel length
`rad_symmetry`	`symm`	kernel symmetry
`unsigned int`	`data_len`	data input length
`rad_support_region`	`support`	output region of support
`unsigned int`	`out_len`	output length
`unsigned int`	`decimation`	output decimation factor

Return Value

none.

Description

Returns the attributes for a 1D convolution object.

Restrictions
Errors

The arguments must conform to the following:

1. The 1D convolution object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

The length of the kernel is also known as the filter order.

rad_convolve1d_f

Compute a decimated real one-dimensional (1D) convolution of two vectors.

Prototype

```
void rad_convolve1d_f(
                  rad_conv1d_f  *plan,
                  float         *x,
                  rad_stride     stridex,
                  float         *y,
                  rad_stride     stridey,
                  unsigned int   n);
```

Parameters

`plan`, structure, input.

`x`, real vector, input.

`stridex`, integer scalar, input.

`y`, real vector, output.

`stridey`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

Uses a 1D convolution object to compute the convolution of a real filter (kernel) vector h of length M with a real data vector x of length N and output decimation factor D to produce an output vector y.
See `rad_conv1d_create_f` for details.

Restrictions
Errors

The arguments must conform to the following:

1. The 1D convolution object `plan` must be valid.
2. The input vector `x` must be of length N (conformant with the 1D convolution object).
3. The output vector `y` must be of length (conformant with the 1D convolution object):

   Full: $\lfloor (N + M - 2)/D \rfloor + 1$
   Same: $\lfloor (N - 1)/D \rfloor + 1$
   Minimum: $\lfloor (N - 1)/D \rfloor - \lfloor (M - 1)/D \rfloor + 1$.

4. The input `x`, and the output `y`, must not overlap.

Notes

The decimation factor, D, is normally one for non-lowpass filters.
If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments.

rad_conv2d_create_f

Create a decimated 2D convolution filter object.

Prototype

```
rad_conv2d_f * rad_conv2d_create_f(
                   float               *H,
                   rad_stride           ldH,
                   rad_symmetry         symm,
                   unsigned int         P,
                   unsigned int         Q,
                   unsigned int         D,
                   rad_support_region   support,
                   unsigned int         ntimes,
                   rad_alg_hint         hint,
                   unsigned int         m,
                   unsigned int         n);
```

Parameters

H, real matrix, input. Matrix of non-redundant filter coefficients. There are M by N in the non-symmetric case and $\lceil M/2 \rceil$ by $\lceil N/2 \rceil$ for symmetric filters.

ldH, integer scalar, input.

symm, enumerated type, input.

RAD_NONSYM	non-symmetric
RAD_SYM_EVEN_LEN_ODD	(even) symmetric, odd length
RAD_SYM_EVEN_LEN_EVEN	(even) symmetric, even length

P, integer scalar, input. Number of rows in data matrix.

Q, integer scalar, input. Number of columns in data matrix.

D, integer scalar, input. Decimation factor.

support, enumerated type, input.

RAD_SUPPORT_FULL	maximum region
RAD_SUPPORT_SAME	input and output same size
RAD_SUPPORT_MIN	region without zero extending the kernel

ntimes, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.

hint, enumerated type, input.

RAD_ALG_SPACE	minimise memory usage
RAD_ALG_TIME	minimise execution time
RAD_ALG_NOISE	maximise numerical accuracy

m, integer scalar, input.

n, integer scalar, input.

Return Value

structure.

Description

Creates a decimated 2D convolution filter object and returns a pointer to the object. The user specifies the kernel (filter order, symmetry, and filter coefficients), the region of support, and the integral output decimation factor.
If the create fails, `NULL` is returned.
A 2D convolution object is used to compute the convolution of a real filter (kernel) matrix H of size M by N with a real data matrix X of size P by Q, producing the output matrix Y. The filter size must be less than or equal to the size of the data.

Full:
size $= \lfloor (P+M-2)/D \rfloor + 1$ by $\lfloor (Q+N-2)/D \rfloor + 1$
$y[j,k] := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h[m,n] \langle x[jD-m, kD-n] \rangle$ for $0 \leqslant j \leqslant \lfloor (P+M-2)/D \rfloor$ and $0 \leqslant k \leqslant \lfloor (Q+N-2)/D \rfloor$.

Same size:
length $= \lfloor (P-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor + 1$
$y[j,k] := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h[m,n] \langle x[jD + \lfloor M/2 \rfloor - m, kD + \lfloor N/2 \rfloor - n] \rangle$ for $0 \leqslant j \leqslant \lfloor (P-1)/D \rfloor$ and $0 \leqslant k \leqslant \lfloor (Q-1)/D \rfloor$.

Minimum (not zero padded):
length $= \lfloor (P-1)/D \rfloor - \lfloor (M-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor - \lfloor (N-1)/D \rfloor + 1$.
$y[j,k] := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} h[m,n] \langle x[jD + (M-1) - m, kD + (N-1) - n] \rangle$ for $0 \leqslant j \leqslant \lfloor (P-1)/D \rfloor - \lfloor (M-1)/D \rfloor$ and $0 \leqslant k \leqslant \lfloor (Q-1)/D \rfloor - \lfloor (N-1)/D \rfloor$.

Where $\langle x[j] \rangle = \begin{cases} x[j,k] & : \ 0 \leqslant j < P \text{ and } 0 \leqslant k < Q \\ 0 & : \ \text{otherwise.} \end{cases}$
The filter kernel can be even–symmetric or non-symmetric. If it is symmetric the redundant values are not specified.

Restrictions

The filter length must be less than or equal to the data length: $M \leqslant P$ and $N \leqslant Q$.
Memory major order must be the same for kernel, data and output.
The kernel, data, and output matrix must be unit stride in the major direction.

Errors

Notes

The symmetry, support and decimation attributes apply uniformly to all dimensions.

rad_conv2d_destroy_f

Destroy a 2D convolution object.

Prototype

```
int rad_conv2d_destroy_f(
                    rad_conv2d_f *plan);
```

Parameters

`plan`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) a 2D convolution object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The arguments must conform to the following:

1. The 2D convolution object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_conv2d_getattr_f

Returns the attributes for a 2D convolution object.

Prototype

```
void rad_conv2d_getattr_f(
                  rad_conv2d_f      *plan,
                  rad_conv2d_attr_f *attr);
```

Parameters

`plan`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`rad_scalar_mi`	`kernel_size`	kernel size
`rad_symmetry`	`symm`	kernel symmetry
`rad_scalar_mi`	`in_size`	data input size
`rad_support_region`	`support`	output region of support
`rad_scalar_mi`	`out_size`	output size
`unsigned int`	`decimation`	output decimation factor

Return Value

none.

Description

Returns the attributes for a 2D convolution object.

Restrictions
Errors

The arguments must conform to the following:

1. The 2D convolution object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

The size of the kernel is also known as the filter order.

rad_convolve2d_f

Compute a decimated real two-dimensional (2D) convolution of two matrices.

Prototype

```
void rad_convolve2d_f(
                  rad_conv2d_f *plan,
                  float        *x,
                  rad_stride    ldx,
                  float        *y,
                  rad_stride    ldy,
                  unsigned int  m,
                  unsigned int  n);
```

Parameters

plan, structure, input.

x, real matrix, size m by n, input.

ldx, integer scalar, input.

y, real matrix, size p by q, output.

ldy, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

Uses a 2D convolution object to compute the convolution of a real filter (kernel) matrix H of size M by N with a real data matrix X of size P by Q, producing the output matrix Y. The filter size must be less than or equal to the size of the data.
See rad_conv2d_create_f for details.

Restrictions

Memory major order must be the same for kernel, data and output. The kernel, data, and output matrix are restricted to unit stride in the major direction.

Errors

The arguments must conform to the following:

1. The 2D convolution object plan must be valid.
2. The input matrix x must be of size P by Q (conformant with the 2D convolution object).
3. The output matrix y must be of size (conformant with the 2D convolution object):

   Full: $\lfloor (P+M-2)/D \rfloor + 1$ by $\lfloor (Q+N-2)/D \rfloor + 1$
   Same: $\lfloor (P-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor + 1$
   Minimum: $\lfloor (P-1)/D \rfloor - \lfloor (M-1)/D \rfloor + 1$ by $\lfloor (Q-1)/D \rfloor - \lfloor (N-1)/D \rfloor + 1$.

4. The input x, and the output y, must not overlap.

Notes

The decimation factor, D, is normally one for non-lowpass filters.
If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments.

rad_Dcorr1d_create_P

Create a 1D correlation object.

Prototype

```
rad_Dcorr1d_P * rad_Dcorr1d_create_P(
                  unsigned int       M,
                  unsigned int       N,
                  rad_support_region support,
                  unsigned int       ntimes,
                  rad_alg_hint       hint);
```

The following instances are supported:

rad_corr1d_create_f

rad_ccorr1d_create_f

Parameters

M, integer scalar, input. Length of input reference vector.

N, integer scalar, input. Length of input data vector.

support, enumerated type, input.

RAD_SUPPORT_FULL	maximum region
RAD_SUPPORT_SAME	input and output same size
RAD_SUPPORT_MIN	region without zero extending the kernel

Output region of support (indicates which output points are computed).

ntimes, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.

hint, enumerated type, input.

RAD_ALG_SPACE	minimise memory usage
RAD_ALG_TIME	minimise execution time
RAD_ALG_NOISE	maximise numerical accuracy

Return Value

structure.

Description

Creates a (cross-) correlation object and returns a pointer to the object. The user specifies the lengths of the reference vector r and the data vector x.
If the create fails, NULL is returned.
A 1D correlation object is used to compute the (cross-) correlation of a reference vector r of length M with a data vector x of length N to produce an output vector y.

Full:
length $= N + M - 1$

$$\widehat{y}[k] := \sum_{j=0}^{M-1} r[j]\,\langle x[k+j-M+1]^*\rangle \qquad \text{for } 0 \leqslant k \leqslant N+M-2$$

$$y[k] := \widehat{y}[k] * \begin{cases} 1/(k+1) & : \quad 0 \leqslant k < M-1 \\ 1/M & : \quad M-1 \leqslant k < N \\ 1/(N+M-1-k) & : \quad N \leqslant k < N+M-1. \end{cases}$$

Same size:
length $= N$

$$\widehat{y}[k] := \sum_{j=0}^{M-1} r[j]\langle x[k+j-\lfloor M/2 \rfloor]^* \rangle \qquad \text{for } 0 \leqslant k \leqslant N-1$$

$$y[k] := \widehat{y}[k] * \begin{cases} 1/(k+\lceil M/2 \rceil) & : \quad 0 \leqslant k < \lfloor M/2 \rfloor \\ 1/M & : \quad \lfloor M/2 \rfloor \leqslant k < N - \lfloor M/2 \rfloor \\ 1/(N+\lfloor M/2 \rfloor - k) & : \quad N - \lfloor M/2 \rfloor \leqslant k < N. \end{cases}$$

Minimum (not zero padded):
length $= N - M + 1$

$$\widehat{y}[k] := \sum_{j=0}^{M-1} r[j]\langle x[k+j]^* \rangle \qquad \text{for } 0 \leqslant k \leqslant N-M$$

$$y[k] := \widehat{y}[k]/M.$$

$$\text{Where } \langle x[j] \rangle = \begin{cases} x[j] & : \quad 0 \leqslant j < N \\ 0 & : \quad \text{otherwise.} \end{cases}$$

The values $\widehat{y}[k]$ are the biased correlation estimates while the $y[k]$ are unbiased estimates. (The unbiased estimates are scaled by the number or terms in the summation for each lag where $\langle x[j] \rangle$ is not defined to be zero.)

Restrictions

The reference length must be less than or equal to the data length, $M \leqslant N$.

Errors

The arguments must conform to the following:

1. $1 \leqslant$ `M` $\leqslant$ `N`.
2. `support` must be valid.
3. `hint` must be valid.

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments: specify the FIR kernel as the reverse-indexed clone of the reference data.

rad_Dcorr1d_destroy_P

Destroy a 1D correlation object.

Prototype

```
int rad_Dcorr1d_destroy_P(
                    rad_Dcorr1d_P *plan);
```

The following instances are supported:

rad_corr1d_destroy_f

rad_ccorr1d_destroy_f

Parameters

plan, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) a 1D correlation object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The arguments must conform to the following:

1. The 1D correlation object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

rad_Dcorr1d_getattr_P

Return the attributes for a 1D correlation object.

Prototype

```
void rad_Dcorr1d_getattr_P(
                  rad_Dcorr1d_P      *plan,
                  rad_Dcorr1d_attr_P *attr);
```

The following instances are supported:

rad_corr1d_getattr_f

rad_ccorr1d_getattr_f

Parameters

plan, structure, input.

attr, pointer to structure, output.
The attribute structure contains the following information:

unsigned int	ref_len	reference length
unsigned int	data_len	data input length
rad_support_region	support	output region of support
unsigned int	lag_len	output (lags) length

Return Value

none.

Description

Returns the attributes for a 1D correlation object.

Restrictions

Errors

The arguments must conform to the following:

1. The 1D correlation object plan must be valid.
2. The attribute pointer attr must not be NULL.

Notes

rad_Dcorrelate1d_P

Compute a real one-dimensional (1D) correlation of two vectors.

Prototype

```
void rad_Dcorrelate1d_P(
                  rad_Dcorr1d_P      *plan,
                  rad_bias            bias,
                  rad_scalar_P/void  *ref,
                  rad_stride          strideref,
                  rad_scalar_P/void  *x,
                  rad_stride          stridex,
                  rad_scalar_P/void  *y,
                  rad_stride          stridey,
                  unsigned int        n);
```

The following instances are supported:

`rad_correlate1d_f`

`rad_ccorrelate1d_f`

Parameters

`plan`, structure, input.

`bias`, enumerated type, input.

`RAD_BIASED`	biased
`RAD_UNBIASED`	unbiased

Return biased or unbiased estimates.

`ref`, real or complex vector, input.

`strideref`, integer scalar, input.

`x`, real or complex vector, input.

`stridex`, integer scalar, input.

`y`, real or complex vector, output.

`stridey`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

Uses a 1D correlation object to compute the (cross-) correlation of a reference vector r of length M with a data vector x of length N to produce an output vector y.
See `rad_Dcorr1d_create_P` for details.

Restrictions

The reference length must be less than or equal to the data length, $M \leqslant N$.

Errors

The arguments must conform to the following:

1. The 1D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input vector `ref` must be of length M (conformant with the 1D correlation object).
4. The data input vector `x` must be of length N (conformant with the 1D correlation object).
5. The output vector `y` must be of length (conformant with the 1D correlation object):

   Full: $N + M - 1$

   Same: N

   Minimum: $N - M + 1$.
6. The output `y` cannot overlap either of the input vector, `ref` or `x`.

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments: specify the FIR kernel as the reverse-indexed clone of the reference data.

rad_ccorrelate1d_split_f

Compute a real one-dimensional (1D) correlation of two vectors.

Prototype

```
void rad_ccorrelate1d_split_f(
                  rad_ccorr1d_f *plan,
                  rad_bias       bias,
                  float         *ref_re,
                  float         *ref_im,
                  rad_stride     strideref,
                  float         *x_re,
                  float         *x_im,
                  rad_stride     stridex,
                  float         *y_re,
                  float         *y_im,
                  rad_stride     stridey,
                  unsigned int   n);
```

Parameters

plan, structure, input.

bias, enumerated type, input.

RAD_BIASED	biased
RAD_UNBIASED	unbiased

Return biased or unbiased estimates.

ref_re, real part of complex vector, length m, input.

ref_im, imaginary part of complex vector, length m, input.

strideref, integer scalar, input.

x_re, real part of complex vector, length n, input.

x_im, imaginary part of complex vector, length n, input.

stridex, integer scalar, input.

y_re, real part of complex vector, length p, output.

y_im, imaginary part of complex vector, length p, output.

stridey, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

Uses a 1D correlation object to compute the (cross-) correlation of a reference vector r of length M with a data vector x of length N to produce an output vector y.
See rad_ccorr1d_create_f for details.

Restrictions

The reference length must be less than or equal to the data length, $M \leqslant N$.

Errors

The arguments must conform to the following:

1. The 1D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input vector `ref` must be of length M (conformant with the 1D correlation object).
4. The data input vector `x` must be of length N (conformant with the 1D correlation object).
5. The output vector `y` must be of length (conformant with the 1D correlation object):

   Full: $N + M - 1$

   Same: N

   Minimum: $N - M + 1$.
6. The output `y` cannot overlap either of the input vector, `ref` or `x`.

Notes

If all of the data are not available at one time, use the FIR filtering routines to filter the data in segments: specify the FIR kernel as the reverse-indexed clone of the reference data.

rad_Dcorr2d_create_P

Create a 2D correlation object.

Prototype

```
rad_Dcorr2d_P * rad_Dcorr2d_create_P(
                  unsigned int       M,
                  unsigned int       N,
                  unsigned int       P,
                  unsigned int       Q,
                  rad_support_region support,
                  unsigned int       ntimes,
                  rad_alg_hint       hint);
```

The following instances are supported:

rad_corr2d_create_f

rad_ccorr2d_create_f

Parameters

M, integer scalar, input. Number of rows in reference matrix.

N, integer scalar, input. Number of columns in reference matrix.

P, integer scalar, input. Number of rows in data matrix.

Q, integer scalar, input. Number of columns in data matrix.

support, enumerated type, input.

RAD_SUPPORT_FULL	maximum region
RAD_SUPPORT_SAME	input and output same size
RAD_SUPPORT_MIN	region without zero extending the kernel

Output region of support (indicates which output points are computed).

ntimes, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.

hint, enumerated type, input.

RAD_ALG_SPACE	minimise memory usage
RAD_ALG_TIME	minimise execution time
RAD_ALG_NOISE	maximise numerical accuracy

Return Value

structure.

Description

Creates a (cross-) correlation object and returns a pointer to the object. The user specifies the sizes of the reference matrix R and the data matrix X.
If the create fails, NULL is returned.
A 2D correlation object is used to compute the (cross-) correlation of a reference matrix R of size M by N with a data matrix X of size P by Q to produce an output matrix Y.

Full:
size $= P + M - 1$ by $Q + N - 1$

$$\widehat{Y}[j,k] := \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} R[m,n]\,\langle X[m+j-M+1, n+k-N+1]^*\rangle \qquad \text{for } 0 \leqslant j \leqslant P+M-2$$
and $0 \leqslant k \leqslant Q+N-2$

$$Y[j,k] := \widehat{Y}[j,k] * \begin{cases} 1/(j+1) & : \quad 0 \leqslant j < M-1 \\ 1/M & : \quad M-1 \leqslant j < P \\ 1/(P+M-1-i) & : \quad P \leqslant j < P+M-1; \\ 1/(k+1) & : \quad 0 \leqslant k < N-1 \\ 1/N & : \quad N-1 \leqslant k < Q \\ 1/(Q+N-1-k) & : \quad Q \leqslant k < Q+N-1. \end{cases}$$

Same size:
size $= P$ by Q

$$\widehat{Y}[j,k] := \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} R[j,k]\langle X[m+j-\lfloor M/2\rfloor, n+j-\lfloor N/2\rfloor]^*\rangle \qquad \text{for } 0 \leqslant j \leqslant P-1 \text{ and}$$
$0 \leqslant k \leqslant Q-1$

$$Y[j,k] := \widehat{Y}[j,k] * \begin{cases} 1/(j+\lceil M/2\rceil) & : \quad 0 \leqslant j < \lfloor M/2\rfloor \\ 1/M & : \quad \lfloor M/2\rfloor \leqslant j < P-\lfloor M/2\rfloor \\ 1/(P+\lceil M/2\rceil-1-j) & : \quad P-\lfloor M/2\rfloor \leqslant j < P; \\ 1/(k+\lceil N/2\rceil) & : \quad 0 \leqslant k < \lfloor N/2\rfloor \\ 1/N & : \quad \lfloor N/2\rfloor \leqslant k < Q-\lfloor N/2\rfloor \\ 1/(Q+\lceil N/2\rceil-1-k) & : \quad Q-\lfloor N/2\rfloor \leqslant k < Q. \end{cases}$$

Minimum (not zero padded):
size $= P-M+1$ by $Q-N+1$

$$\widehat{Y}[j,k] := \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} R[j,k]\langle X[m+j, n+k]^*\rangle \qquad \text{for } 0 \leqslant j \leqslant P-M \text{ and } 0 \leqslant k \leqslant Q-N$$

$$Y[k] := \widehat{Y}[j,k]/(M\,N).$$

Where $\langle x[j]\rangle = \begin{cases} x[j] & : \quad 0 \leqslant j < N \\ 0 & : \quad \text{otherwise.} \end{cases}$

The values $\widehat{Y}[j,k]$ are the biased correlation estimates while the $Y[j,k]$ are unbiased estimates. (The unbiased estimates are scaled by the number or terms in the summation for each lag where $\langle X[j,k]\rangle$ is not defined to be zero.)

Restrictions

The reference size must be less than or equal to the data size: $M \leqslant P$ and $N \leqslant Q$.

Errors

The arguments must conform to the following:

1. $1 \leqslant$ `M` $\leqslant$ `P` and $1 \leqslant$ `N` $\leqslant$ `Q`.
2. `support` must be valid.
3. `hint` must be valid.

Notes

The support attribute applies uniformly to all dimensions.

rad_Dcorr2d_destroy_P

Destroy a 2D correlation object.

Prototype

```
int rad_Dcorr2d_destroy_P(
                    rad_Dcorr2d_P *plan);
```

The following instances are supported:

`rad_corr2d_destroy_f`

`rad_ccorr2d_destroy_f`

Parameters

`plan`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) a 2D correlation object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The arguments must conform to the following:

1. The 2D correlation object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_Dcorr2d_getattr_P

Return the attributes for a 2D correlation object.

Prototype

```
void rad_Dcorr2d_getattr_P(
                  rad_Dcorr2d_P      *plan,
                  rad_Dcorr2d_attr_P *attr);
```

The following instances are supported:

rad_corr2d_getattr_f

rad_ccorr2d_getattr_f

Parameters

plan, structure, input.

attr, pointer to structure, output.
The attribute structure contains the following information:

rad_scalar_mi	ref_size	reference size
rad_scalar_mi	data_size	data input size
rad_support_region	support	output region of support
rad_scalar_mi	out_size	output size

Return Value

none.

Description

Returns the attributes for a 2D correlation object.

Restrictions
Errors

The arguments must conform to the following:

1. The 2D correlation object plan must be valid.
2. The attribute pointer attr must not be NULL.

Notes

rad_Dcorrelate2d_P

Compute a two-dimensional (2D) correlation of two matrices.

Prototype

```
void rad_Dcorrelate2d_P(
                  rad_Dcorr2d_P      *plan,
                  rad_bias            bias,
                  rad_scalar_P/void  *ref,
                  rad_stride          ldref,
                  rad_scalar_P/void  *x,
                  rad_stride          ldx,
                  rad_scalar_P/void  *y,
                  rad_stride          ldy,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

`rad_correlate2d_f`

`rad_ccorrelate2d_f`

Parameters

`plan`, structure, input.

`bias`, enumerated type, input.

`RAD_BIASED`	biased
`RAD_UNBIASED`	unbiased

Return biased or unbiased estimates.

`ref`, real or complex matrix, input.

`ldref`, integer scalar, input.

`x`, real or complex matrix, input.

`ldx`, integer scalar, input.

`y`, real or complex matrix, output.

`ldy`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

Uses a 2D correlation object to compute the (cross-) correlation of a reference matrix R with a data matrix X to produce an output matrix Y.
See `rad_corr2d_create_f` for details.

Restrictions

The reference size must be less than or equal to the data size: $M \leqslant P$ and $N \leqslant Q$. Memory major order must be the same for reference, data and output. The matrix views must be unit stride in the major direction.

Errors

The arguments must conform to the following:

1. The 2D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input matrix `ref` must be of size M by N (conformant with the 2D correlation object).
4. The data input matrix `x` must be of size P by Q (conformant with the 2D correlation object).
5. The output matrix `y` must be of size (conformant with the 2D correlation object):

   Full: $P + M - 1$ by $Q + N - 1$

   Same: P by Q

   Minimum: $P - M + 1$ by $Q - N + 1$.
6. The output `y` cannot overlap either of the input matrices, `ref` or `x`.
7. Memory major order must be the same for reference, data and output.
8. The matrix views must be unit stride in the major direction.

Notes

rad_ccorrelate2d_split_f

Compute a two-dimensional (2D) correlation of two matrices.

Prototype

```
void rad_ccorrelate2d_split_f(
                  rad_ccorr2d_f *plan,
                  rad_bias       bias,
                  float         *ref_re,
                  float         *ref_im,
                  rad_stride     ldref,
                  float         *x_re,
                  float         *x_im,
                  rad_stride     ldx,
                  float         *y_re,
                  float         *y_im,
                  rad_stride     ldy,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

plan, structure, input.

bias, enumerated type, input.

RAD_BIASED	biased
RAD_UNBIASED	unbiased

Return biased or unbiased estimates.

ref_re, real part of complex matrix, size m by n, input.

ref_im, imaginary part of complex matrix, size m by n, input.

ldref, integer scalar, input.

x_re, real part of complex matrix, size p by q, input.

x_im, imaginary part of complex matrix, size p by q, input.

ldx, integer scalar, input.

y_re, real part of complex matrix, size P by Q, output.

y_im, imaginary part of complex matrix, size P by Q, output.

ldy, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

Uses a 2D correlation object to compute the (cross-) correlation of a reference matrix R with a data matrix X to produce an output matrix Y.
See rad_ccorr2d_create_f for details.

Restrictions

The reference size must be less than or equal to the data size: $M \leqslant P$ and $N \leqslant Q$. Memory major order must be the same for reference, data and output. The matrix views must be unit stride in the major direction.

Errors

The arguments must conform to the following:

1. The 2D correlation object `plan` must be valid.
2. `bias` must be valid.
3. The reference input matrix `ref` must be of size M by N (conformant with the 2D correlation object).
4. The data input matrix `x` must be of size P by Q (conformant with the 2D correlation object).
5. The output matrix `y` must be of size (conformant with the 2D correlation object):

   Full: $P + M - 1$ by $Q + N - 1$

   Same: P by Q

   Minimum: $P - M + 1$ by $Q - N + 1$.
6. The output `y` cannot overlap either of the input matrices, `ref` or `x`.
7. Memory major order must be the same for reference, data and output.
8. The matrix views must be unit stride in the major direction.

Notes

7.3 Window Functions

rad_vcreate_blackman_f

rad_vcreate_cheby_f

rad_vcreate_hanning_f

rad_vcreate_kaiser_f

rad_free

rad_vcreate_blackman_f

Create a vector with Blackman window weights.

Prototype

```
float  * rad_vcreate_blackman_f(
                  unsigned int    N,
                  rad_memory_hint hint);
```

Parameters

N, integer scalar, input. Length of window.

hint, enumerated type, input.

RAD_MEM_NONE	no hint
RAD_MEM_RDONLY	read-only
RAD_MEM_CONST	constant
RAD_MEM_SHARED	shared
RAD_MEM_SHARED_RDONLY	shared and read-only
RAD_MEM_SHARED_CONST	shared and constant

Return Value

real vector.

Description

Creates a vector initialised with a Blackman window of length N.
$X[k] := 0.42 - 0.5\cos\left(\frac{2\pi k}{N-1}\right) + 0.8\cos\left(\frac{4\pi k}{N-1}\right)$.
NULL is returned if the create fails.

Restrictions
Errors

The arguments must conform to the following:

1. N > 1.
2. hint must be valid.

Notes

rad_vcreate_cheby_f

Create a vector with Dolph-Chebyshev window weights.

Prototype

```
float  * rad_vcreate_cheby_f(
                  unsigned int    N,
                  float           ripple,
                  rad_memory_hint hint);
```

Parameters

`N`, integer scalar, input. Length of window.

`ripple`, real scalar, input. Side lobes are this number of decibels below the main lobe.

`hint`, enumerated type, input.

`RAD_MEM_NONE`	no hint
`RAD_MEM_RDONLY`	read-only
`RAD_MEM_CONST`	constant
`RAD_MEM_SHARED`	shared
`RAD_MEM_SHARED_RDONLY`	shared and read-only
`RAD_MEM_SHARED_CONST`	shared and constant

Return Value

real vector.

Description

Creates a vector initialised with a Dolph-Chebyshev window of length N.

$\delta_p := 10^{-ripple/20}$

$\tau_p := \dfrac{1+\delta_p}{\delta_p}$

$\delta_f := \dfrac{1}{\pi}\ \cos^{-1}\left[\dfrac{1}{\cosh\left(\frac{\cosh^{-1}(\tau_p)}{N-1}\right)}\right]$

$x[0] := \dfrac{3-\cos(2\pi\delta_f)}{1+\cos(2\pi\delta_f)}$

$x[k] := \dfrac{x[0]+1}{2}\ \left(\cos\left(\dfrac{2\pi k}{N}\right)\right) + \dfrac{x[0]-1}{2}$

N odd:

$$W[k] := \begin{cases} \delta_p\ \cosh\left(\frac{N-1}{2}\ \cosh^{-1}(x[k])\right) & : \quad |x[k]| > 1 \\ \delta_p\ \cos\left(\frac{N-1}{2}\ \cos^{-1}(x[k])\right) & : \quad |x[k]| \leqslant 1 \end{cases}$$

N even:

$$W[k] := \begin{cases} \delta_p\ \cosh\left(\frac{N-1}{2}\ \cosh^{-1}(x[k])\right)\ \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| > 1, \quad 0 \leqslant k \leqslant \lfloor N/2 \rfloor \\ -\delta_p\ \cosh\left(\frac{N-1}{2}\ \cosh^{-1}(x[k])\right)\ \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| > 1, \quad \lfloor N/2 \rfloor < k < N \\ \delta_p\ \cos\left(\frac{N-1}{2}\ \cos^{-1}(x[k])\right)\ \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| \leqslant 1, \quad 0 \leqslant k \leqslant \lfloor N/2 \rfloor \\ -\delta_p\ \cos\left(\frac{N-1}{2}\ \cos^{-1}(x[k])\right)\ \exp(\pi \mathrm{i} k/N) & : \quad |x[k]| \leqslant 1, \quad \lfloor N/2 \rfloor < k < N. \end{cases}$$

FFT the W's:

$$w[k] := \sum_{j=0}^{N-1} W[k]\ \exp(2\pi \mathrm{i} jk/N).$$

Populate the window with the frequency swap of the w's:

$$X[k] := \begin{cases} \text{real}\left(\frac{w[k+\lfloor N/2 \rfloor]}{w[0]}\right) & : \quad 0 \leqslant k < \lfloor N/2 \rfloor \\ \text{real}\left(\frac{w[k-\lfloor N/2 \rfloor]}{w[0]}\right) & : \quad \lfloor N/2 \rfloor \leqslant k < N. \end{cases}$$

`NULL` is returned if the create fails.

Restrictions

Errors

The arguments must conform to the following:

1. `N` > 0.
2. `hint` must be a valid.

Notes

rad_vcreate_hanning_f

Create a vector with Hanning window weights.

Prototype

```
float  * rad_vcreate_hanning_f(
                    unsigned int    N,
                    rad_memory_hint hint);
```

Parameters

`N`, integer scalar, input. Length of window.

`hint`, enumerated type, input.

`RAD_MEM_NONE`	no hint
`RAD_MEM_RDONLY`	read-only
`RAD_MEM_CONST`	constant
`RAD_MEM_SHARED`	shared
`RAD_MEM_SHARED_RDONLY`	shared and read-only
`RAD_MEM_SHARED_CONST`	shared and constant

Return Value

real vector.

Description

Creates a vector initialised with a Hanning window of length N.
$X[k] := 0.5\left(1 - \cos\left(\frac{2\pi(k+1)}{N+1}\right)\right)$.
`NULL` is returned if the create fails.

Restrictions

Restrictions

Errors

The arguments must conform to the following:

1. `N` > 1.
2. `hint` must be valid.

Notes

There are two different widely used definitions of the Hanning window. The other is
$X[k] := 0.5\left(1 - \cos\left(\frac{2\pi k)}{N-1}\right)\right)$.
This form has a weight of zero for both end points of the window; we use the form that does not have zero end points.
If you want the window to be periodic of length N, you must generate a Hanning window of length $N-1$, copy it to a vector of length N, and set the last point to 0.0.

rad_vcreate_kaiser_f

Create a vector with Kaiser window weights.

Prototype

```
float  * rad_vcreate_kaiser_f(
                    unsigned int    N,
                    float           beta,
                    rad_memory_hint hint);
```

Parameters

N, integer scalar, input. Length of window.

beta, real scalar, input. Transition width.

hint, enumerated type, input.

RAD_MEM_NONE	no hint
RAD_MEM_RDONLY	read-only
RAD_MEM_CONST	constant
RAD_MEM_SHARED	shared
RAD_MEM_SHARED_RDONLY	shared and read-only
RAD_MEM_SHARED_CONST	shared and constant

Return Value

real vector.

Description

Creates a vector initialised with a Kaiser window of length N.

$$X[k] := \frac{I_0\left[\beta\sqrt{1-\left(\frac{2k-N+1}{N-1}\right)^2}\right]}{I_0[\beta]} \text{ where } I_0[x] = \sum_{p=0}^{\infty}\left(\frac{x^p}{2^p\,p!}\right)^2.$$

Increasing β widens the main lobe (transition width) and reduces the side lobes.
NULL is returned if the create fails.

Restrictions

Errors

Notes

rad_free

Releases the memory allocated for the window weights.

Prototype

```
void rad_free(
                    void *ptr);
```

Parameters

`ptr`, pointer to structure, input.

Return Value

none.

Description

Releases the memory allocated by the `rad_vcreate_*_f()` functions.

Restrictions

This function may only be used to release memory allocated by a call to `rad_vcreate_*_f()`.

Errors

Notes

Under Linux, this function is synonymous with `free()`. Microsoft Windows, however, has a special function to generate aligned memory blocks which must be matched in the DLL with the corresponding `free()`. If you try to `free()` the pointer returned by `rad_vcreate_*_f()` the results may be unpredictable.

7.4 Filter Functions

rad_Dfir_create_P

Create a decimated FIR filter object.

Prototype

```
rad_Dfir_P * rad_Dfir_create_P(
                  rad_scalar_P/void  *kernel,
                  rad_stride          stridekernel,
                  rad_symmetry        symm,
                  unsigned int        N,
                  unsigned int        D,
                  rad_obj_state       state,
                  unsigned int        ntimes,
                  rad_alg_hint        hint,
                  unsigned int        n);
```

The following instances are supported:

`rad_fir_create_f`

`rad_cfir_create_f`

Parameters

`kernel`, real or complex vector, input. Vector of non-redundant filter coefficients. There are $M+1$ in the non-symmetric case and $\lceil (M+1)/2 \rceil$ for symmetric filters.

`stridekernel`, integer scalar, input.

`symm`, enumerated type, input.

`RAD_NONSYM`	non-symmetric
`RAD_SYM_EVEN_LEN_ODD`	(even) symmetric, odd length
`RAD_SYM_EVEN_LEN_EVEN`	(even) symmetric, even length

`N`, integer scalar, input. Length of data vector.

`D`, integer scalar, input. Decimation factor.

`state`, enumerated type, input.

`RAD_STATE_NO_SAVE`	do not save state — single call filter
`RAD_STATE_SAVE`	save state for continuous filter

`ntimes`, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.

`hint`, enumerated type, input.

`RAD_ALG_SPACE`	minimise memory usage
`RAD_ALG_TIME`	minimise execution time
`RAD_ALG_NOISE`	maximise numerical accuracy

`n`, integer scalar, input.

Return Value

structure.

Description

Creates a decimated FIR filter object and returns a pointer to the object. The user specifies the kernel (filter coefficients and filter order), the integral output decimation factor, D, the length of the input segments (vectors) that will be filtered, and whether to save state information for continuous filtering.

If the create fails, `NULL` is returned.

If requested, the FIR filter object encapsulates the filter's state information. The state is initialised to zero. The filter state allows long data streams to be processed in segments by successive calls to `rad_Dfirflt_P`.

The FIR filter object is used to compute:

$$y[k] := \sum_{j=0}^{M} h[j]\ \widehat{x}[p + kD - j] \text{ where } \widehat{x}[j] = \begin{cases} s[j] & : \quad j < 0 \\ x[j] & : \quad j \geqslant 0 \end{cases}, \text{ and } 0 \leqslant k < \lceil (N-p)/D \rceil.$$

The vector s and integer p are private internal state information. When the FIR filter object is created they are initialised to zero, and they will remain so if the SAVE option is not specified. Otherwise

$s[j] := x[N + j]$ for $-M \leqslant j < 0$ and

$p := D - 1 - [(N - 1 - p) \bmod D]$.

Given a filter kernel of order M with coefficient vector h, segment length N, and decimation factor D, the decimated output y is of length $(N - p)/D$.

Restrictions

The decimation factor must be less than or equal to the filter length.

Errors

Notes

For non-lowpass filters, set $D = 1$.

It is important that the kernel vector be only as long as necessary (see above) — the symmetric values of the filter between the kernel's centre and its last value are not to be included in the kernel.

It is safe to destroy the kernel after creating the FIR filter object.

The filter will be evaluated directly unless `hint` is `RAD_ALG_TIME`, in which case convolution in the frequency domain (a method based on FFTs) will be used if it is quicker.

rad_cfir_create_split_f

Create a decimated FIR filter object.

Prototype

```
rad_cfir_f * rad_cfir_create_split_f(
                    float          *kernel_re,
                    float          *kernel_im,
                    rad_stride      stridekernel,
                    rad_symmetry    symm,
                    unsigned int    N,
                    unsigned int    D,
                    rad_obj_state   state,
                    unsigned int    ntimes,
                    rad_alg_hint    hint,
                    unsigned int    n);
```

Parameters

kernel_re, real part of complex vector, length n, input.

kernel_im, imaginary part of complex vector, length n, input. Vector of non-redundant filter coefficients. There are $M + 1$ in the non-symmetric case and $\lceil (M + 1)/2 \rceil$ for symmetric filters.

stridekernel, integer scalar, input.

symm, enumerated type, input.

RAD_NONSYM	non-symmetric
RAD_SYM_EVEN_LEN_ODD	(even) symmetric, odd length
RAD_SYM_EVEN_LEN_EVEN	(even) symmetric, even length

N, integer scalar, input. Length of data vector.

D, integer scalar, input. Decimation factor.

state, enumerated type, input.

RAD_STATE_NO_SAVE	do not save state — single call filter
RAD_STATE_SAVE	save state for continuous filter

ntimes, integer scalar, input. An estimate of how many times the filter will be used. Zero is treated as 'many'.

hint, enumerated type, input.

RAD_ALG_SPACE	minimise memory usage
RAD_ALG_TIME	minimise execution time
RAD_ALG_NOISE	maximise numerical accuracy

n, integer scalar, input.

Return Value

structure.

Description

Creates a decimated FIR filter object and returns a pointer to the object. The user specifies the kernel (filter coefficients and filter order), the integral output decimation factor, D, the length

of the input segments (vectors) that will be filtered, and whether to save state information for continuous filtering.

If the create fails, NULL is returned.

If requested, the FIR filter object encapsulates the filter's state information. The state is initialised to zero. The filter state allows long data streams to be processed in segments by successive calls to rad_cfirflt_f.

The FIR filter object is used to compute:

$$y[k] := \sum_{j=0}^{M} h[j]\ \widehat{x}[p + kD - j] \text{ where } \widehat{x}[j] = \begin{cases} s[j] & : \quad j < 0 \\ x[j] & : \quad j \geqslant 0 \end{cases}, \text{ and } 0 \leqslant k < \lceil (N-p)/D \rceil.$$

The vector s and integer p are private internal state information. When the FIR filter object is created they are initialised to zero, and they will remain so if the SAVE option is not specified. Otherwise

$s[j] := x[N + j]$ for $-M \leqslant j < 0$ and

$p := D - 1 - [(N - 1 - p) \bmod D]$.

Given a filter kernel of order M with coefficient vector h, segment length N, and decimation factor D, the decimated output y is of length $(N - p)/D$.

Restrictions

The decimation factor must be less than or equal to the filter length.

Errors

Notes

For non-lowpass filters, set $D = 1$.

It is important that the kernel vector be only as long as necessary (see above) — the symmetric values of the filter between the kernel's centre and its last value are not to be included in the kernel.

It is safe to destroy the kernel after creating the FIR filter object.

The filter will be evaluated directly unless hint is RAD_ALG_TIME, in which case convolution in the frequency domain (a method based on FFTs) will be used if it is quicker.

rad_Dfir_destroy_P

Destroy a FIR filter object.

Prototype

```
int rad_Dfir_destroy_P(
                    rad_Dfir_P *plan);
```

The following instances are supported:

`rad_fir_destroy_f`

`rad_cfir_destroy_f`

Parameters

`plan`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) a FIR filter object. Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

An argument of `NULL` is not an error.

rad_Dfirflt_P

FIR filter an input sequence and decimate the output.

Prototype

```
int rad_Dfirflt_P(
                  rad_Dfir_P          *plan,
                  rad_scalar_P/void   *x,
                  rad_stride           stridex,
                  rad_scalar_P/void   *y,
                  rad_stride           stridey,
                  unsigned int         n);
```

The following instances are supported:

`rad_firflt_f`

`rad_cfirflt_f`

Parameters

`plan`, structure, input.

`x`, real or complex vector, input.

`stridex`, integer scalar, input.

`y`, real or complex vector, output.

`stridey`, integer scalar, input.

`n`, integer scalar, input.

Return Value

number of values computed.

Description

Applies a FIR filter, specified by the FIR filter object, to an input segment x, and computes a decimated output segment y. Initial and final filter state is encapsulated in the FIR filter object. Long data streams can be processed in segments by successive calls to this function.
When N is a multiple of D, the length of y and the number of output samples is N/D.
When N is not a multiple of D, the length of y is $\lceil N/D \rceil$, although it may not be fully populated; there may be only $\lfloor N/D \rfloor$ values.
The return value is the number of output samples computed.

Restrictions

Filtering cannot be performed in place.

Errors

The arguments must conform to the following:

1. The FIR filter object must be valid.
2. The input vector `x` must be of length N (conformant with the FIR filter object).
3. The output vector `y` must be of length $\lceil N/D \rceil$ (conformant with the FIR filter object).
4. The input `x`, and the output `y`, must not overlap.

Notes

The filter object may be modified with the updated state.

rad_cfirflt_split_f

FIR filter an input sequence and decimate the output.

Prototype

```
int rad_cfirflt_split_f(
                    rad_cfir_f    *plan,
                    float         *x_re,
                    float         *x_im,
                    rad_stride     stridex,
                    float         *y_re,
                    float         *y_im,
                    rad_stride     stridey,
                    unsigned int   n);
```

Parameters

`plan`, structure, input.

`x_re`, real part of complex vector, length n, input.

`x_im`, imaginary part of complex vector, length n, input.

`stridex`, integer scalar, input.

`y_re`, real part of complex vector, length m, output.

`y_im`, imaginary part of complex vector, length m, output.

`stridey`, integer scalar, input.

`n`, integer scalar, input.

Return Value

number of values computed.

Description

Applies a FIR filter, specified by the FIR filter object, to an input segment x, and computes a decimated output segment y. Initial and final filter state is encapsulated in the FIR filter object. Long data streams can be processed in segments by successive calls to this function.
When N is a multiple of D, the length of y and the number of output samples is N/D.
When N is not a multiple of D, the length of y is $\lceil N/D \rceil$, although it may not be fully populated; there may be only $\lfloor N/D \rfloor$ values.
The return value is the number of output samples computed.

Restrictions

Filtering cannot be performed in place.

Errors

The arguments must conform to the following:

1. The FIR filter object must be valid.
2. The input vector `x` must be of length N (conformant with the FIR filter object).
3. The output vector `y` must be of length $\lceil N/D \rceil$ (conformant with the FIR filter object).
4. The input `x`, and the output `y`, must not overlap.

Notes

The filter object may be modified with the updated state.

rad_Dfir_getattr_P

Return the attributes of a FIR filter object.

Prototype

```
void rad_Dfir_getattr_P(
                  rad_Dfir_P      *plan,
                  rad_Dfir_attr_P *attr);
```

The following instances are supported:

`rad_fir_getattr_f`

`rad_cfir_getattr_f`

Parameters

`plan`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`unsigned int`	`kernel_len`	kernel length
`rad_symmetry`	`symm`	kernel symmetry
`unsigned int`	`in_len`	filter input segment length
`unsigned int`	`out_len`	filter output segment length
`unsigned int`	`decimation`	decimation factor
`rad_obj_state`	`state`	save state information

Return Value

none.

Description

Returns the attributes of a FIR filter object.

Restrictions
Errors

The arguments must conform to the following:

1. The filter object `plan` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

The filter coefficient values are not accessible attributes.

rad_fir_reset_f

Reset the state of a decimated FIR filter object.

Prototype

```
void rad_fir_reset_f(
                    rad_fir_f *fir);
```

Parameters

`fir`, structure, input.

Return Value

none.

Description

Resets the internal state of a previously created FIR filter object to the same state it had immediately after creation.

Restrictions
Errors

The arguments must conform to the following:

1. The filter object must be valid.

Notes

7.5 Miscellaneous Signal Processing Functions

rad_vhisto_f

rad_vhisto_f

Compute the histogram of a vector.

Prototype

```
void rad_vhisto_f(
                  float         *A,
                  rad_stride     strideA,
                  float          min,
                  float          max,
                  rad_hist_opt   opt,
                  float         *R,
                  rad_stride     strideR,
                  unsigned int   n);
```

Parameters

A, real vector, length n, input.

strideA, integer scalar, input.

min, real scalar, input.

max, real scalar, input.

opt, enumerated type, input.

RAD_HIST_RESET	reset histogram each time
RAD_HIST_ACCUM	accumulate histogram

R, real vector, output.

strideR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

Computes the histogram of a vector. Suppose the number of bins in the output vector is P. The first and last elements of the output vector are used to accumulate values outside the range of interest. The bin size is determined from the remaining $P-2$ bins and the boundary values.
The output vector is initialised to zero if the RESET option is used, otherwise the histogram is accumulated on top of the current data in the output vector.
The bin b is assigned as follows: if $\mathtt{A}[j*\mathtt{strideA}] < \mathtt{min}$ then $b := 0$ else if $\mathtt{A}[j*\mathtt{strideA}] \geqslant \mathtt{max}$ then $b := P-1$ else $b := 1 + \lfloor (P-2)(\mathtt{A}[j*\mathtt{strideA}] - \mathtt{min})/(\mathtt{max}-\mathtt{min}) \rfloor$.

Restrictions

Errors

The arguments must conform to the following:

1. All the vector objects must be valid and of positive length.
2. $\mathtt{min} < \mathtt{max}$.

Notes

The first and last bins collect all the values less than min, and greater or equal to max, respectively.

8 • Linear Algebra

8.1 Matrix and Vector Operations

rad_vouter_f

rad_cvouter_f

rad_cvouter_split_f

rad_vcsummgval_f

rad_vcsummgval_split_f

rad_Dminvlu_P

rad_cminvlu_split_f

rad_cmherm_f

Complex Hermitian (conjugate transpose) of a matrix.

Prototype

```
void rad_cmherm_f(
                  void          *A,
                  rad_stride     ldA,
                  void          *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n);
```

Parameters

A, complex matrix, size m by n, input.

ldA, integer scalar, input.

R, complex matrix, size n by m, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R := A^H.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

rad_cmherm_split_f

Complex Hermitian (conjugate transpose) of a matrix.

Prototype

```
void rad_cmherm_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of complex matrix, size n by m, output.

R_im, imaginary part of complex matrix, size n by m, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$R := A^H$.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

rad_cvjdot_f

Compute the conjugate inner (dot) product of two complex vectors.

Prototype

```
rad_cscalar_f rad_cvjdot_f(
                    void          *A,
                    rad_stride     strideA,
                    void          *B,
                    rad_stride     strideB,
                    unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

B, complex vector, length n, input.

strideB, integer scalar, input.

n, integer scalar, input.

Return Value

complex scalar.

Description

return value := $A^T \cdot B^*$.

Restrictions

Overflow may occur.

Errors

Notes

rad_cvjdot_split_f

Compute the conjugate inner (dot) product of two complex vectors.

Prototype

```
rad_cscalar_f rad_cvjdot_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

n, integer scalar, input.

Return Value

complex scalar.

Description

return value := $\mathrm{A}^{\mathrm{T}} \cdot \mathrm{B}^{*}$.

Restrictions

Overflow may occur.

Errors

Notes

rad_gemp_f

Calculate the general product of two matrices and accumulate.

Prototype

```
void rad_gemp_f(
                    float          alpha,
                    float         *A,
                    rad_stride     ldA,
                    rad_mat_op     Aop,
                    float         *B,
                    rad_stride     ldB,
                    rad_mat_op     Bop,
                    float          beta,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n,
                    unsigned int   p);
```

Parameters

alpha, real scalar, input.

A, real matrix, size m by p, input. The size shows the dimensions after Aop has been applied.

ldA, integer scalar, input.

Aop, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose

B, real matrix, size p by n, input. The size shows the dimensions after Bop has been applied.

ldB, integer scalar, input.

Bop, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose

beta, real scalar, input.

R, real matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$\mathtt{R} := \mathtt{alpha} \cdot \mathtt{Aop}(\mathtt{A}) \cdot \mathtt{Bop}(\mathtt{B}) + \mathtt{beta} \cdot \mathtt{R}$
where Aop and Bop are matrix operations
(identity or transpose).

Restrictions
Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` and `Bop` must be valid.

Notes

rad_cgemp_f

Calculate the general product of two matrices and accumulate.

Prototype

```
void rad_cgemp_f(
                  void          *alpha,
                  void          *A,
                  rad_stride     ldA,
                  rad_mat_op     Aop,
                  void          *B,
                  rad_stride     ldB,
                  rad_mat_op     Bop,
                  void          *beta,
                  void          *R,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n,
                  unsigned int   p);
```

Parameters

alpha, complex scalar, input.

A, complex matrix, size m by p, input. The size shows the dimensions after Aop has been applied.

ldA, integer scalar, input.

Aop, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)
RAD_MAT_CONJ	conjugate

B, complex matrix, size p by n, input. The size shows the dimensions after Bop has been applied.

ldB, integer scalar, input.

Bop, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)
RAD_MAT_CONJ	conjugate

beta, complex scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$\texttt{R} := \texttt{alpha} \cdot \texttt{Aop}(\texttt{A}) \cdot \texttt{Bop}(\texttt{B}) + \texttt{beta} \cdot \texttt{R}$
where `Aop` and `Bop` are matrix operations (identity, transpose, conjugate or Hermitian).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` and `Bop` must be valid.

Notes

rad_cgemp_split_f

Calculate the general product of two matrices and accumulate.

Prototype

```
void rad_cgemp_split_f(
                  float         *alpha_re,
                  float         *alpha_im,
                  float         *A_re,
                  float         *A_im,
                  rad_stride     ldA,
                  rad_mat_op     Aop,
                  float         *B_re,
                  float         *B_im,
                  rad_stride     ldB,
                  rad_mat_op     Bop,
                  float         *beta_re,
                  float         *beta_im,
                  float         *R_re,
                  float         *R_im,
                  rad_stride     ldR,
                  unsigned int   m,
                  unsigned int   n,
                  unsigned int   p);
```

Parameters

alpha_re, real part of complex scalar, input.

alpha_im, imaginary part of complex scalar, input.

A_re, real part of complex matrix, size m by p, input.

A_im, imaginary part of complex matrix, size m by p, input. The size shows the dimensions after Aop has been applied.

ldA, integer scalar, input.

Aop, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)
RAD_MAT_CONJ	conjugate

B_re, real part of complex matrix, size p by n, input.

B_im, imaginary part of complex matrix, size p by n, input. The size shows the dimensions after Bop has been applied.

ldB, integer scalar, input.

Bop, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)
RAD_MAT_CONJ	conjugate

beta_re, real part of complex scalar, input.

beta_im, imaginary part of complex scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$\texttt{R} := \texttt{alpha} \cdot \texttt{Aop}(\texttt{A}) \cdot \texttt{Bop}(\texttt{B}) + \texttt{beta} \cdot \texttt{R}$
where Aop and Bop are matrix operations
(identity, transpose, conjugate or Hermitian).

Restrictions

Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. Aop and Bop must be valid.

Notes

rad_gems_f

Calculate a general matrix sum.

Prototype

```
void rad_gems_f(
                    float          alpha,
                    float         *A,
                    rad_stride     ldA,
                    rad_mat_op     Aop,
                    float          beta,
                    float         *C,
                    rad_stride     ldC,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

`alpha`, real scalar, input.

`A`, real matrix, size m by n, input. The size shows the dimensions after `Aop` has been applied.

`ldA`, integer scalar, input.

`Aop`, enumerated type, input.

`RAD_MAT_NTRANS`	no transformation
`RAD_MAT_TRANS`	transpose

`beta`, real scalar, input.

`C`, real matrix, size m by n, output.

`ldC`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{C} := \texttt{alpha} \cdot \texttt{Aop}(\texttt{A}) + \texttt{beta} \cdot \texttt{C}$
where `Aop` is a matrix operation (identity or transpose).

Restrictions
Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` must be valid.

Notes

rad_cgems_f

Calculate a general matrix sum.

Prototype

```
void rad_cgems_f(
                    void          *alpha,
                    void          *A,
                    rad_stride     ldA,
                    rad_mat_op     Aop,
                    void          *beta,
                    void          *C,
                    rad_stride     ldC,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

`alpha`, complex scalar, input.

`A`, complex matrix, size m by n, input. The size shows the dimensions after `Aop` has been applied.

`ldA`, integer scalar, input.

`Aop`, enumerated type, input.

`RAD_MAT_NTRANS`	no transformation
`RAD_MAT_TRANS`	transpose
`RAD_MAT_HERM`	Hermitian (conjugate transpose)
`RAD_MAT_CONJ`	conjugate

`beta`, complex scalar, input.

`C`, complex matrix, size m by n, output.

`ldC`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

$\texttt{C} := \texttt{alpha} \cdot \texttt{Aop}(\texttt{A}) + \texttt{beta} \cdot \texttt{C}$
where `Aop` is a matrix operation (identity, transpose, conjugate or Hermitian).

Restrictions
Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` must be valid.

Notes

rad_cgems_split_f

Calculate a general matrix sum.

Prototype

```
void rad_cgems_split_f(
                    float         *alpha_re,
                    float         *alpha_im,
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    rad_mat_op     Aop,
                    float         *beta_re,
                    float         *beta_im,
                    float         *C_re,
                    float         *C_im,
                    rad_stride     ldC,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

alpha_re, real part of complex scalar, input.

alpha_im, imaginary part of complex scalar, input.

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input. The size shows the dimensions after Aop has been applied.

ldA, integer scalar, input.

Aop, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)
RAD_MAT_CONJ	conjugate

beta_re, real part of complex scalar, input.

beta_im, imaginary part of complex scalar, input.

C_re, real part of complex matrix, size m by n, output.

C_im, imaginary part of complex matrix, size m by n, output.

ldC, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$\mathtt{C} := \mathtt{alpha} \cdot \mathtt{Aop}(\mathtt{A}) + \mathtt{beta} \cdot \mathtt{C}$
where Aop is a matrix operation (identity, transpose, conjugate or Hermitian).

Restrictions
Errors

The arguments must conform to the following:

1. The matrices must be conformant.
2. `Aop` must be valid.

Notes

rad_Dmprod_P

Calculate the product of two matrices.

Prototype

```
void rad_Dmprod_P(
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  rad_scalar_P/void  *B,
                  rad_stride          ldB,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n,
                  unsigned int        p);
```

The following instances are supported:

- rad_mprod_f
- rad_mprod_i
- rad_mprod_si
- rad_cmprod_f
- rad_cmprod_i
- rad_cmprod_si

Parameters

A, real or complex matrix, size m by p, input.

ldA, integer scalar, input.

B, real or complex matrix, size p by n, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$R := A \cdot B$.

Restrictions

Errors

Notes

rad_Dmprod_split_P

Calculate the product of two matrices.

Prototype

```
void rad_Dmprod_split_P (
                    rad_scalar_P *A_re,
                    rad_scalar_P *A_im,
                    rad_stride    ldA,
                    rad_scalar_P *B_re,
                    rad_scalar_P *B_im,
                    rad_stride    ldB,
                    rad_scalar_P *R_re,
                    rad_scalar_P *R_im,
                    rad_stride    ldR,
                    unsigned int  m,
                    unsigned int  n,
                    unsigned int  p);
```

The following instances are supported:

rad_cmprod_split_f

rad_cmprod_split_i

rad_cmprod_split_si

Parameters

A_re, real part of real or complex matrix, size m by p, input.

A_im, imaginary part of real or complex matrix, size m by p, input.

ldA, integer scalar, input.

B_re, real part of real or complex matrix, size p by n, input.

B_im, imaginary part of real or complex matrix, size p by n, input.

ldB, integer scalar, input.

R_re, real part of real or complex matrix, size m by n, output.

R_im, imaginary part of real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

R := A · B.

Restrictions

Errors

Notes

rad_cmprodh_P

Calculate the product a complex matrix and the Hermitian of a complex matrix.

Prototype

```
void rad_cmprodh_P (
                    rad_scalar_P  *A,
                    rad_stride     ldA,
                    rad_scalar_P  *B,
                    rad_stride     ldB,
                    rad_scalar_P  *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n,
                    unsigned int   p);
```

The following instances are supported:

rad_cmprodh_f

rad_cmprodh_i

rad_cmprodh_si

Parameters

A, complex matrix, size m by p, input.

ldA, integer scalar, input.

B, complex matrix, size n by p, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$R := A \cdot B^H$.

Restrictions

Errors

Notes

rad_cmprodh_split_P

Calculate the product a complex matrix and the Hermitian of a complex matrix.

Prototype

```
void rad_cmprodh_split_P(
                    rad_scalar_P  *A_re,
                    rad_scalar_P  *A_im,
                    rad_stride     ldA,
                    rad_scalar_P  *B_re,
                    rad_scalar_P  *B_im,
                    rad_stride     ldB,
                    rad_scalar_P  *R_re,
                    rad_scalar_P  *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n,
                    unsigned int   p);
```

The following instances are supported:

rad_cmprodh_split_f

rad_cmprodh_split_i

rad_cmprodh_split_si

Parameters

A_re, real part of complex matrix, size m by p, input.

A_im, imaginary part of complex matrix, size m by p, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size n by p, input.

B_im, imaginary part of complex matrix, size n by p, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$R := A \cdot B^H$.

Restrictions

Errors

Notes

rad_cmprodj_P

Calculate the product a complex matrix and the conjugate of a complex matrix.

Prototype

```
void rad_cmprodj_P(
                   rad_scalar_P  *A,
                   rad_stride     ldA,
                   rad_scalar_P  *B,
                   rad_stride     ldB,
                   rad_scalar_P  *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n,
                   unsigned int   p);
```

The following instances are supported:

rad_cmprodj_f

rad_cmprodj_i

rad_cmprodj_si

Parameters

A, complex matrix, size m by p, input.

ldA, integer scalar, input.

B, complex matrix, size p by n, input.

ldB, integer scalar, input.

R, complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$R := A \cdot B^*$.

Restrictions

Errors

Notes

rad_cmprodj_split_P

Calculate the product a complex matrix and the conjugate of a complex matrix.

Prototype

```
void rad_cmprodj_split_P(
                    rad_scalar_P  *A_re,
                    rad_scalar_P  *A_im,
                    rad_stride     ldA,
                    rad_scalar_P  *B_re,
                    rad_scalar_P  *B_im,
                    rad_stride     ldB,
                    rad_scalar_P  *R_re,
                    rad_scalar_P  *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n,
                    unsigned int   p);
```

The following instances are supported:

rad_cmprodj_split_f

rad_cmprodj_split_i

rad_cmprodj_split_si

Parameters

A_re, real part of complex matrix, size m by p, input.

A_im, imaginary part of complex matrix, size m by p, input.

ldA, integer scalar, input.

B_re, real part of complex matrix, size p by n, input.

B_im, imaginary part of complex matrix, size p by n, input.

ldB, integer scalar, input.

R_re, real part of complex matrix, size m by n, output.

R_im, imaginary part of complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

R := A · B^*.

Restrictions

Errors

Notes

rad_Dmprodt_P

Calculate the product of a matrix and the transpose of a matrix.

Prototype

```
void rad_Dmprodt_P (
                    rad_scalar_P/void  *A,
                    rad_stride          ldA,
                    rad_scalar_P/void  *B,
                    rad_stride          ldB,
                    rad_scalar_P/void  *R,
                    rad_stride          ldR,
                    unsigned int        m,
                    unsigned int        n,
                    unsigned int        p);
```

The following instances are supported:

rad_mprodt_f

rad_mprodt_i

rad_mprodt_si

rad_cmprodt_f

rad_cmprodt_i

rad_cmprodt_si

Parameters

A, real or complex matrix, size m by p, input.

ldA, integer scalar, input.

B, real or complex matrix, size n by p, input.

ldB, integer scalar, input.

R, real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$R := A \cdot B^T$.

Restrictions

Errors

Notes

rad_Dmprodt_split_P

Calculate the product of a matrix and the transpose of a matrix.

Prototype

```
void rad_Dmprodt_split_P(
                    rad_scalar_P *A_re,
                    rad_scalar_P *A_im,
                    rad_stride    ldA,
                    rad_scalar_P *B_re,
                    rad_scalar_P *B_im,
                    rad_stride    ldB,
                    rad_scalar_P *R_re,
                    rad_scalar_P *R_im,
                    rad_stride    ldR,
                    unsigned int  m,
                    unsigned int  n,
                    unsigned int  p);
```

The following instances are supported:

rad_cmprodt_split_f

rad_cmprodt_split_i

rad_cmprodt_split_si

Parameters

A_re, real part of real or complex matrix, size m by p, input.

A_im, imaginary part of real or complex matrix, size m by p, input.

ldA, integer scalar, input.

B_re, real part of real or complex matrix, size n by p, input.

B_im, imaginary part of real or complex matrix, size n by p, input.

ldB, integer scalar, input.

R_re, real part of real or complex matrix, size m by n, output.

R_im, imaginary part of real or complex matrix, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

none.

Description

$R := A \cdot B^T$.

Restrictions

Errors

Notes

rad_Dmvprod_P

Calculate a matrix–vector product.

Prototype

```
void rad_Dmvprod_P (
                    rad_scalar_P/void  *A,
                    rad_stride          ldA,
                    rad_scalar_P/void  *X,
                    rad_stride          strideX,
                    rad_scalar_P/void  *Y,
                    rad_stride          strideY,
                    unsigned int        m,
                    unsigned int        n);
```

The following instances are supported:

- rad_mvprod_f
- rad_mvprod_i
- rad_mvprod_si
- rad_cmvprod_f
- rad_cmvprod_i
- rad_cmvprod_si

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

X, real or complex vector, length n, input.

strideX, integer scalar, input.

Y, real or complex vector, length m, output.

strideY, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$Y := A \cdot X$.

Restrictions

Errors

Notes

rad_Dmvprod_split_P

Calculate a matrix–vector product.

Prototype

```
void rad_Dmvprod_split_P(
                    rad_scalar_P *A_re,
                    rad_scalar_P *A_im,
                    rad_stride    ldA,
                    rad_scalar_P *X_re,
                    rad_scalar_P *X_im,
                    rad_stride    strideX,
                    rad_scalar_P *Y_re,
                    rad_scalar_P *Y_im,
                    rad_stride    strideY,
                    unsigned int  m,
                    unsigned int  n);
```

The following instances are supported:

rad_cmvprod_split_f

rad_cmvprod_split_i

rad_cmvprod_split_si

Parameters

A_re, real part of real or complex matrix, size m by n, input.

A_im, imaginary part of real or complex matrix, size m by n, input.

ldA, integer scalar, input.

X_re, real part of real or complex vector, length n, input.

X_im, imaginary part of real or complex vector, length n, input.

strideX, integer scalar, input.

Y_re, real part of real or complex vector, length m, output.

Y_im, imaginary part of real or complex vector, length m, output.

strideY, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$Y := A \cdot X$.

Restrictions

Errors

Notes

rad_Dmtrans_P

Transpose a matrix.

Prototype

```
void rad_Dmtrans_P(
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        m,
                  unsigned int        n);
```

The following instances are supported:

- rad_mtrans_bl
- rad_mtrans_f
- rad_mtrans_i
- rad_mtrans_si
- rad_cmtrans_f
- rad_cmtrans_i
- rad_cmtrans_si

Parameters

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

R, real or complex matrix, size n by m, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$R := A^T$.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

rad_Dmtrans_split_P

Transpose a matrix.

Prototype

```
void rad_Dmtrans_split_P(
                  rad_scalar_P *A_re,
                  rad_scalar_P *A_im,
                  rad_stride    ldA,
                  rad_scalar_P *R_re,
                  rad_scalar_P *R_im,
                  rad_stride    ldR,
                  unsigned int  m,
                  unsigned int  n);
```

The following instances are supported:

rad_cmtrans_split_f

rad_cmtrans_split_i

rad_cmtrans_split_si

Parameters

A_re, real part of real or complex matrix, size m by n, input.

A_im, imaginary part of real or complex matrix, size m by n, input.

ldA, integer scalar, input.

R_re, real part of real or complex matrix, size n by m, output.

R_im, imaginary part of real or complex matrix, size n by m, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$R := A^T$.

Restrictions

If the matrix A is square, the transpose is in place if A and R resolve to the same object, otherwise A and R must be disjoint.

Errors

Notes

rad_Dvdot_P

Compute the inner (dot) product of two vectors.

Prototype

```
rad_Dscalar_P rad_Dvdot_P(
                  rad_scalar_P/void  *A,
                  rad_stride          strideA,
                  rad_scalar_P/void  *B,
                  rad_stride          strideB,
                  unsigned int        n);
```

The following instances are supported:

`rad_vdot_f`

`rad_cvdot_f`

Parameters

`A`, real or complex vector, length n, input.

`strideA`, integer scalar, input.

`B`, real or complex vector, length n, input.

`strideB`, integer scalar, input.

`n`, integer scalar, input.

Return Value

real or complex scalar.

Description

return value := $\mathtt{A}^{\mathrm{T}} \cdot \mathtt{B}$.

Restrictions

Overflow may occur.

Errors

Notes

rad_cvdot_split_f

Compute the inner (dot) product of two vectors.

Prototype

```
rad_cscalar_f rad_cvdot_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     strideA,
                    float         *B_re,
                    float         *B_im,
                    rad_stride     strideB,
                    unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input.

strideB, integer scalar, input.

n, integer scalar, input.

Return Value

complex scalar.

Description

return value := $A^T \cdot B$.

Restrictions

Overflow may occur.

Errors

Notes

rad_Dvmprod_P

Calculate a vector–matrix product.

Prototype

```
void rad_Dvmprod_P (
                    rad_scalar_P/void  *X,
                    rad_stride          strideX,
                    rad_scalar_P/void  *A,
                    rad_stride          ldA,
                    rad_scalar_P/void  *Y,
                    rad_stride          strideY,
                    unsigned int        m,
                    unsigned int        n);
```

The following instances are supported:

- rad_vmprod_f
- rad_vmprod_i
- rad_vmprod_si
- rad_cvmprod_f
- rad_cvmprod_i
- rad_cvmprod_si

Parameters

X, real or complex vector, length m, input.

strideX, integer scalar, input.

A, real or complex matrix, size m by n, input.

ldA, integer scalar, input.

Y, real or complex vector, length n, output.

strideY, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$Y := X^T \cdot A$.

Restrictions

Errors

Notes

rad_Dvmprod_split_P

Calculate a vector–matrix product.

Prototype

```
void rad_Dvmprod_split_P(
                    rad_scalar_P *X_re,
                    rad_scalar_P *X_im,
                    rad_stride    strideX,
                    rad_scalar_P *A_re,
                    rad_scalar_P *A_im,
                    rad_stride    ldA,
                    rad_scalar_P *Y_re,
                    rad_scalar_P *Y_im,
                    rad_stride    strideY,
                    unsigned int  m,
                    unsigned int  n);
```

The following instances are supported:

rad_cvmprod_split_f

rad_cvmprod_split_i

rad_cvmprod_split_si

Parameters

X_re, real part of real or complex vector, length m, input.

X_im, imaginary part of real or complex vector, length m, input.

strideX, integer scalar, input.

A_re, real part of real or complex matrix, size m by n, input.

A_im, imaginary part of real or complex matrix, size m by n, input.

ldA, integer scalar, input.

Y_re, real part of real or complex vector, length n, output.

Y_im, imaginary part of real or complex vector, length n, output.

strideY, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

$Y := X^T \cdot A$.

Restrictions

Errors

Notes

rad_vouter_f

Calculate the outer product of two vectors.

Prototype

```
void rad_vouter_f(
                    float          alpha,
                    float         *X,
                    rad_stride     strideX,
                    float         *Y,
                    rad_stride     strideY,
                    float         *R,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

`alpha`, real scalar, input.

`X`, real vector, length m, input.

`strideX`, integer scalar, input.

`Y`, real vector, length n, input.

`strideY`, integer scalar, input.

`R`, real vector, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R` := `alpha` $\cdot$ `X` $\cdot$ `Y`$^{\mathrm{T}}$.

Restrictions

Errors

Notes

rad_cvouter_f

Calculate the outer product of two vectors.

Prototype

```
void rad_cvouter_f(
                   void          *alpha,
                   void          *X,
                   rad_stride     strideX,
                   void          *Y,
                   rad_stride     strideY,
                   void          *R,
                   rad_stride     ldR,
                   unsigned int   m,
                   unsigned int   n);
```

Parameters

`alpha`, complex scalar, input.

`X`, complex vector, length m, input.

`strideX`, integer scalar, input.

`Y`, complex vector, length n, input.

`strideY`, integer scalar, input.

`R`, complex vector, size m by n, output.

`ldR`, integer scalar, input.

`m`, integer scalar, input.

`n`, integer scalar, input.

Return Value

none.

Description

`R` := `alpha` $\cdot$ `X` $\cdot$ `Y`^{H}.

Restrictions

Errors

Notes

rad_cvouter_split_f

Calculate the outer product of two vectors.

Prototype

```
void rad_cvouter_split_f(
                    float         *alpha_re,
                    float         *alpha_im,
                    float         *X_re,
                    float         *X_im,
                    rad_stride     strideX,
                    float         *Y_re,
                    float         *Y_im,
                    rad_stride     strideY,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   m,
                    unsigned int   n);
```

Parameters

alpha_re, real part of complex scalar, input.

alpha_im, imaginary part of complex scalar, input.

X_re, real part of complex vector, length m, input.

X_im, imaginary part of complex vector, length m, input.

strideX, integer scalar, input.

Y_re, real part of complex vector, length n, input.

Y_im, imaginary part of complex vector, length n, input.

strideY, integer scalar, input.

R_re, real part of complex vector, size m by n, output.

R_im, imaginary part of complex vector, size m by n, output.

ldR, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R := alpha · X · Y^H.

Restrictions

Errors

Notes

rad_vcsummgval_f

Returns the sum of the magnitudes of the elements of a complex vector.

Prototype

```
float  rad_vcsummgval_f(
                   void          *A,
                   rad_stride     strideA,
                   unsigned int   n);
```

Parameters

A, complex vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

real scalar.

Description

return value $:= \sum |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

rad_vcsummgval_split_f

Returns the sum of the magnitudes of the elements of a complex vector.

Prototype

```
float  rad_vcsummgval_split_f(
                   float         *A_re,
                   float         *A_im,
                   rad_stride     strideA,
                   unsigned int   n);
```

Parameters

A_re, real part of complex vector, length n, input.

A_im, imaginary part of complex vector, length n, input.

strideA, integer scalar, input.

n, integer scalar, input.

Return Value

real scalar.

Description

return value $:= \sum |\mathtt{A}[j * \mathtt{strideA}]|$ where $0 \leqslant j < n$.

Restrictions

Errors

Notes

The order of summation is not specified, therefore significant numerical errors may occur.

rad_Dminvlu_P

Invert a square matrix using LU decomposition.

Prototype

```
void rad_Dminvlu_P (
                  rad_scalar_P/void  *A,
                  rad_stride          ldA,
                  signed int         *V,
                  rad_stride          strideV,
                  rad_scalar_P/void  *R,
                  rad_stride          ldR,
                  unsigned int        n);
```

The following instances are supported:

rad_minvlu_f

rad_cminvlu_f

Parameters

A, real or complex matrix, size n by n, input.

ldA, integer scalar, input.

V, integer vector, length n, input.

strideV, integer scalar, input.

R, real or complex matrix, size n by n, output.

ldR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R $:=$ A^{-1} where $0 \leqslant j < n$.
V is a list of pivots. It must not be null.
On return, all values will be positive if the inversion is successful; the first element will be -1 if the input matrix is singular or -2 if a memory allocation failed.

Restrictions

Errors

Notes

rad_cminvlu_split_f

Invert a square matrix using LU decomposition.

Prototype

```
void rad_cminvlu_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    signed int    *V,
                    rad_stride     strideV,
                    float         *R_re,
                    float         *R_im,
                    rad_stride     ldR,
                    unsigned int   n);
```

Parameters

A_re, real part of complex matrix, size n by n, input.

A_im, imaginary part of complex matrix, size n by n, input.

ldA, integer scalar, input.

V, integer vector, length n, input.

strideV, integer scalar, input.

R_re, real part of complex matrix, size n by n, output.

R_im, imaginary part of complex matrix, size n by n, output.

ldR, integer scalar, input.

n, integer scalar, input.

Return Value

none.

Description

R $:=$ A^{-1} where $0 \leqslant j < n$.
V is a list of pivots. It must not be null.
On return, all values will be positive if the inversion is successful; the first element will be -1 if the input matrix is singular or -2 if a memory allocation failed.

Restrictions

Errors

Notes

8.2 Special Linear System Solvers

rad_covsol_f

rad_ccovsol_f

rad_ccovsol_split_f

rad_llsqsol_f

rad_cllsqsol_f

rad_cllsqsol_split_f

rad_toepsol_f

rad_ctoepsol_f

rad_ctoepsol_split_f

rad_covsol_f

Solve a covariance linear system problem.

Prototype

```
int rad_covsol_f(
                    float         *A,
                    rad_stride     ldA,
                    float         *XB,
                    rad_stride     ldXB,
                    unsigned int   m,
                    unsigned int   n,
                    unsigned int   p);
```

Parameters

A, real matrix, size m by n, input. The matrix A.

ldA, integer scalar, input.

XB, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: A is not of full rank.

Description

Solves the covariance linear system problem $A^{\mathrm{T}}AX = B$ where A is an m by n matrix of rank n and B is an n by p matrix, and $n \leqslant m$.
Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A may be overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.
The matrix A is assumed to be of full rank. This property is checked and a positive return value indicates that an error occurred.

rad_ccovsol_f

Solve a covariance linear system problem.

Prototype

```
int rad_ccovsol_f(
                  void          *A,
                  rad_stride     ldA,
                  void          *XB,
                  rad_stride     ldXB,
                  unsigned int   m,
                  unsigned int   n,
                  unsigned int   p);
```

Parameters

A, complex matrix, size m by n, input. The matrix A.

ldA, integer scalar, input.

XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: A is not of full rank.

Description

Solves the covariance linear system problem $A^{\mathrm{H}}AX = B$ where A is an m by n matrix of rank n and B is an n by p matrix, and $n \leqslant m$.
Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A may be overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.
The matrix A is assumed to be of full rank. This property is checked and a positive return value indicates that an error occurred.

rad_ccovsol_split_f

Solve a covariance linear system problem.

Prototype

```
int rad_ccovsol_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *XB_re,
                    float         *XB_im,
                    rad_stride     ldXB,
                    unsigned int   m,
                    unsigned int   n,
                    unsigned int   p);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input. The matrix A.

ldA, integer scalar, input.

XB_re, real part of complex matrix, size n by p, modified in place.

XB_im, imaginary part of complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: A is not of full rank.

Description

Solves the covariance linear system problem $A^{\mathrm{H}}AX = B$ where A is an m by n matrix of rank n and B is an n by p matrix, and $n \leqslant m$.
Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A may be overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.
The matrix A is assumed to be of full rank. This property is checked and a positive return value indicates that an error occurred.

rad_llsqsol_f

Solve a linear least squares problem.

Prototype

```
int rad_llsqsol_f(
                  float         *A,
                  rad_stride     ldA,
                  float         *XB,
                  rad_stride     ldXB,
                  unsigned int   m,
                  unsigned int   n,
                  unsigned int   p);
```

Parameters

A, real matrix, size m by n, input. The matrix A.

ldA, integer scalar, input.

XB, real matrix, size m by p, modified in place. On input, the matrix B. On output, the first n rows contain the matrix X.

ldXB, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: A is not of full rank.

Description

Solves the linear least squares problem $\min||AX - B||_2$ where A is an m by n matrix of rank n and B is an m by p matrix, and $n \leqslant m$.
Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A is overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.
The matrix A is assumed to be of full rank. This property is checked. A positive return value indicates that the matrix did not have full column rank and the algorithm failed to be completed.

rad_cllsqsol_f

Solve a linear least squares problem.

Prototype

```
int rad_cllsqsol_f(
                   void          *A,
                   rad_stride     ldA,
                   void          *XB,
                   rad_stride     ldXB,
                   unsigned int   m,
                   unsigned int   n,
                   unsigned int   p);
```

Parameters

A, complex matrix, size m by n, input. The matrix A.

ldA, integer scalar, input.

XB, complex matrix, size m by p, modified in place. On input, the matrix B. On output, the first n rows contain the matrix X.

ldXB, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: A is not of full rank.

Description

Solves the linear least squares problem $\min||AX - B||_2$ where A is an m by n matrix of rank n and B is an m by p matrix, and $n \leqslant m$.
Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A is overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.
The matrix A is assumed to be of full rank. This property is checked. A positive return value indicates that the matrix did not have full column rank and the algorithm failed to be completed.

rad_cllsqsol_split_f

Solve a linear least squares problem.

Prototype

```
int rad_cllsqsol_split_f(
                    float         *A_re,
                    float         *A_im,
                    rad_stride     ldA,
                    float         *XB_re,
                    float         *XB_im,
                    rad_stride     ldXB,
                    unsigned int   m,
                    unsigned int   n,
                    unsigned int   p);
```

Parameters

A_re, real part of complex matrix, size m by n, input.

A_im, imaginary part of complex matrix, size m by n, input. The matrix A.

ldA, integer scalar, input.

XB_re, real part of complex matrix, size m by p, modified in place.

XB_im, imaginary part of complex matrix, size m by p, modified in place. On input, the matrix B. On output, the first n rows contain the matrix X.

ldXB, integer scalar, input.

m, integer scalar, input.

n, integer scalar, input.

p, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: A is not of full rank.

Description

Solves the linear least squares problem $\min||AX - B||_2$ where A is an m by n matrix of rank n and B is an m by p matrix, and $n \leqslant m$.
Returns zero on success, -1 on memory allocation failure, and a positive value if A does not have full column rank.

Restrictions

The matrix A is overwritten.

Errors

Notes

This function allocates and frees its own temporary workspace, which may result in non-deterministic execution time. The more general QR routines may be used to solve a covariance problem and they support explicit creation and destruction.
The matrix A is assumed to be of full rank. This property is checked. A positive return value indicates that the matrix did not have full column rank and the algorithm failed to be completed.

rad_toepsol_f

Solve a real symmetric positive definite Toeplitz linear system.

Prototype

```
int rad_toepsol_f(
                  float         *T,
                  rad_stride     strideT,
                  float         *B,
                  rad_stride     strideB,
                  float         *W,
                  rad_stride     strideW,
                  float         *X,
                  rad_stride     strideX,
                  unsigned int   n);
```

Parameters

`T`, real vector, length n, input. First row of the Toeplitz matrix T.

`strideT`, integer scalar, input.

`B`, real vector, length n, input. The vector B.

`strideB`, integer scalar, input.

`W`, real vector, length n, modified in place. Workspace.

`strideW`, integer scalar, input.

`X`, real vector, length n, output. The vector x.

`strideX`, integer scalar, input.

`n`, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: `T` is not positive definite.

Description

Solves the real symmetric positive definite Toeplitz linear system $Tx = B$ where

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & t_{n-2} & t_{n-1} \\ t_1 & t_0 & t_1 & & t_{n-2} \\ \vdots & t_1 & \ddots & \ddots & \vdots \\ t_{n-2} & & \ddots & \ddots & t_1 \\ t_{n-1} & t_{n-2} & \cdots & t_1 & t_0 \end{bmatrix}.$$

We only need a vector containing the first row of T to specify the system.
Returns zero on success, -1 on memory allocation failure, and a positive value if T is not positive definite.

Restrictions

Errors

Notes

The matrix T is assumed to be of full rank and positive definite. This property is not checked. A positive return value indicates that an error occurred and the algorithm failed to be completed.

rad_ctoepsol_f

Solve a complex Hermitian positive definite Toeplitz linear system.

Prototype

```
int rad_ctoepsol_f(
                   void          *T,
                   rad_stride     strideT,
                   void          *B,
                   rad_stride     strideB,
                   void          *W,
                   rad_stride     strideW,
                   void          *X,
                   rad_stride     strideX,
                   unsigned int   n);
```

Parameters

T, complex vector, length n, input. First row of the Toeplitz matrix T.

strideT, integer scalar, input.

B, complex vector, length n, input. The vector B.

strideB, integer scalar, input.

W, complex vector, length n, modified in place. Workspace.

strideW, integer scalar, input.

X, complex vector, length n, output. The vector x.

strideX, integer scalar, input.

n, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: T is not positive definite.

Description

Solves the complex Hermitian positive definite Toeplitz linear system $Tx = B$ where

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & t_{n-2} & t_{n-1} \\ t_1^* & t_0 & t_1 & & t_{n-2} \\ \vdots & t_1^* & \ddots & \ddots & \vdots \\ t_{n-2}^* & & \ddots & \ddots & t_1 \\ t_{n-1}^* & t_{n-2}^* & \cdots & t_1^* & t_0 \end{bmatrix}.$$

We only need a vector containing the first row of T to specify the system.
Returns zero on success, -1 on memory allocation failure, and a positive value if T is not positive definite.

Restrictions

Errors

Notes

The matrix T is assumed to be of full rank and positive definite. This property is not checked. A positive return value indicates that an error occurred and the algorithm failed to be completed.

rad_ctoepsol_split_f

Solve a complex Hermitian positive definite Toeplitz linear system.

Prototype

```
int rad_ctoepsol_split_f(
                  float         *T_re,
                  float         *T_im,
                  rad_stride     strideT,
                  float         *B_re,
                  float         *B_im,
                  rad_stride     strideB,
                  float         *W_re,
                  float         *W_im,
                  rad_stride     strideW,
                  float         *X_re,
                  float         *X_im,
                  rad_stride     strideX,
                  unsigned int   n);
```

Parameters

T_re, real part of complex vector, length n, input.

T_im, imaginary part of complex vector, length n, input. First row of the Toeplitz matrix T.

strideT, integer scalar, input.

B_re, real part of complex vector, length n, input.

B_im, imaginary part of complex vector, length n, input. The vector B.

strideB, integer scalar, input.

W_re, real part of complex vector, length n, modified in place.

W_im, imaginary part of complex vector, length n, modified in place. Workspace.

strideW, integer scalar, input.

X_re, real part of complex vector, length n, output.

X_im, imaginary part of complex vector, length n, output. The vector x.

strideX, integer scalar, input.

n, integer scalar, input.

Return Value

0 : success.

-1 : out of memory.

$\geqslant 1$: T is not positive definite.

Description

Solves the complex Hermitian positive definite Toeplitz linear system $Tx = B$ where

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & t_{n-2} & t_{n-1} \\ t_1^* & t_0 & t_1 & & t_{n-2} \\ \vdots & t_1^* & \ddots & \ddots & \vdots \\ t_{n-2}^* & & \ddots & \ddots & t_1 \\ t_{n-1}^* & t_{n-2}^* & \cdots & t_1^* & t_0 \end{bmatrix}.$$

We only need a vector containing the first row of T to specify the system.
Returns zero on success, -1 on memory allocation failure, and a positive value if T is not positive definite.

Restrictions

Errors

Notes

The matrix T is assumed to be of full rank and positive definite. This property is not checked. A positive return value indicates that an error occurred and the algorithm failed to be completed.

8.3 General Square Linear System Solver

rad_Dlud_P

rad_clud_split_f

rad_Dlud_create_P

rad_Dlud_destroy_P

rad_Dlud_getattr_P

rad_lusol_f

rad_clusol_f

rad_clusol_split_f

rad_Dlud_P

Compute an LU decomposition of a square matrix using partial pivoting.

Prototype

```
int rad_Dlud_P(
                    rad_clu_P           *lud,
                    rad_scalar_P/void   *A,
                    rad_stride           ldA);
```

The following instances are supported:

`rad_lud_f`

`rad_clud_f`

Parameters

`lud`, structure, input.

`A`, real or complex matrix, size n by n, modified in place.

`ldA`, integer scalar, input.

Return Value

Error code.

Description

Computes the LU decomposition of a general square matrix A using partial pivoting with row interchanges.
An LU decomposition is a factorisation of the form $A = PLU$ where P is a permutation matrix, L is lower triangular, and U is upper triangular.
Returns zero on success, and non-zero if A does not have full rank.

Restrictions

The matrix A is overwritten by the decomposition, and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero pivot element was encountered.

rad_clud_split_f

Compute an LU decomposition of a square matrix using partial pivoting.

Prototype

```
int rad_clud_split_f(
                    rad_clu_f   *lud,
                    float       *A_re,
                    float       *A_im,
                    rad_stride   ldA);
```

Parameters

lud, structure, input.

A_re, real part of complex matrix, size n by n, modified in place.

A_im, imaginary part of complex matrix, size n by n, modified in place.

ldA, integer scalar, input.

Return Value

Error code.

Description

Computes the LU decomposition of a general square matrix A using partial pivoting with row interchanges.
An LU decomposition is a factorisation of the form $A = PLU$ where P is a permutation matrix, L is lower triangular, and U is upper triangular.
Returns zero on success, and non-zero if A does not have full rank.

Restrictions

The matrix A is overwritten by the decomposition, and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero pivot element was encountered.

rad_Dlud_create_P

Create an LU decomposition object.

Prototype

```
rad_Dlu_P * rad_Dlud_create_P(
                    unsigned int  N);
```

The following instances are supported:

`rad_lud_create_f`

`rad_clud_create_f`

Parameters

`N`, integer scalar, input.

Return Value

structure.

Description

Creates an LU decomposition object. The LU decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.
The LU decomposition object is used to compute the LU decomposition of a general square matrix A using partial pivoting with row interchanges.
An LU decomposition is a factorisation of the form $A = PLU$ where P is a permutation matrix, L is lower triangular, and U is upper triangular.
`NULL` is returned if the create fails.

Restrictions

Errors

The input parameter must conform to the following:

1. `N` is positive.

Notes

rad_Dlud_destroy_P

Destroy an LU decomposition object.

Prototype

```
int rad_Dlud_destroy_P(
                    rad_Dlu_P *lud);
```

The following instances are supported:

rad_lud_destroy_f

rad_clud_destroy_f

Parameters

lud, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) an LU decomposition object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The input argument must conform to the following:

1. The LU decomposition object must be valid. An argument of NULL is not an error.

Notes

An argument of NULL is not an error.

rad_Dlud_getattr_P

Returns the attributes of an LU decomposition object.

Prototype

```
void rad_Dlud_getattr_P(
                    rad_Dlu_P       *lud,
                    rad_Dlu_attr_P *attr);
```

The following instances are supported:

`rad_lud_getattr_f`

`rad_clud_getattr_f`

Parameters

`lud`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`unsigned int n` number of rows and columns in matrix

Return Value

none.

Description

Returns the attributes of an LU decomposition object.

Restrictions

Errors

The arguments must conform to the following:

1. The LU decomposition object `lud` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

rad_lusol_f

Solve a square linear system.

Prototype

```
int rad_lusol_f(
                    rad_clu_f     *clud,
                    rad_mat_op     opA,
                    float         *XB,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

clud, structure, input. An LU decomposition object for the matrix A.

opA, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose

XB, real matrix, size n by p, modified in place. (n is implicit, provided by clud.) On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solve the linear system op$(A)X = B$ where op() is either the identity or transpose operation, for a general matrix A using the decomposition computed by rad_lud_f
A is an n by n matrix of rank n and B is an n by p matrix.
Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after rad_lud_f. fails. This will result in a non-zero unsuccessful return value.

rad_clusol_f

Solve a square linear system.

Prototype

```
int rad_clusol_f(
                    rad_clu_f     *clud,
                    rad_mat_op     opA,
                    void          *XB,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

clud, structure, input. An LU decomposition object for the matrix A.

opA, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_HERM	Hermitian (conjugate transpose)

XB, complex matrix, size n by p, modified in place. (n is implicit, provided by clud.) On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solve the linear system op$(A)X = B$ where op() is either the identity or Hermitian operation, for a general matrix A using the decomposition computed by rad_clud_f
A is an n by n matrix of rank n and B is an n by p matrix.
Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after rad_clud_f. fails. This will result in a non-zero unsuccessful return value.

rad_clusol_split_f

Solve a square linear system.

Prototype

```
int rad_clusol_split_f(
                    rad_clu_f     *clud_re,
                    rad_clu_f     *clud_im,
                    rad_mat_op     opA,
                    float         *XB_re,
                    float         *XB_im,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

clud_re, real part of structure, input.

clud_im, imaginary part of structure, input. An LU decomposition object for the matrix A.

opA, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_HERM	Hermitian (conjugate transpose)

XB_re, real part of complex matrix, size n by p, modified in place.

XB_im, imaginary part of complex matrix, size n by p, modified in place. (n is implicit, provided by clud.) On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solve the linear system op$(A)X = B$ where op() is either the identity or Hermitian operation, for a general matrix A using the decomposition computed by rad_clud_f
A is an n by n matrix of rank n and B is an n by p matrix.
Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after rad_clud_f. fails. This will result in a non-zero unsuccessful return value.

8.4 Symmetric Positive Definite Linear System Solver

rad_chold_f

rad_cchold_f

rad_cchold_split_f

rad_chold_create_f

rad_cchold_create_f

rad_Dchold_destroy_P

rad_Dchold_getattr_P

rad_cholsol_f

rad_ccholsol_f

rad_ccholsol_split_f

rad_chold_f

Compute a Cholesky decomposition of a symmetric positive definite matrix.

Prototype

```
int rad_chold_f(
                    rad_cchol_f *chold,
                    float       *A,
                    rad_stride   ldA);
```

Parameters

`chold`, structure, input.

`A`, real matrix, size n by n, modified in place.

`ldA`, integer scalar, input.

Return Value

Error code.

Description

The Cholesky decomposition of a symmetric positive definite n by n matrix A is given by $A = LL^{\mathrm{T}}$ where L is a lower triangular matrix.
There is not a utility function for accessing the factors.
Returns zero on success. The routine will fail if a leading minor is not positive definite.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the decomposition is required.

Errors

The input and input/output objects must conform to the following:

1. All objects must be valid.
2. The matrix `A` and the Cholesky decomposition object `chold` must be conformant.

Notes

The matrix A is assumed to be symmetric. This property is not checked. There is no reduced storage format for symmetric matrices so the full matrix must be specified. However, only half the matrix is referenced and modified.

rad_cchold_f

Compute a Cholesky decomposition of a Hermitian positive definite matrix.

Prototype

```
int rad_cchold_f(
                    rad_cchol_f *chold,
                    void        *A,
                    rad_stride   ldA);
```

Parameters

`chold`, structure, input.

`A`, complex matrix, size n by n, modified in place.

`ldA`, integer scalar, input.

Return Value

Error code.

Description

The Cholesky decomposition of a Hermitian positive definite n by n matrix A is given by $A = LL^{\mathrm{H}}$ where L is a lower triangular matrix.
There is not a utility function for accessing the factors.
Returns zero on success. The routine will fail if a leading minor is not positive definite.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the decomposition is required.

Errors

The input and input/output objects must conform to the following:

1. All objects must be valid.
2. The matrix `A` and the Cholesky decomposition object `chold` must be conformant.

Notes

The matrix A is assumed to be Hermitian. This property is not checked. There is no reduced storage format for Hermitian matrices so the full matrix must be specified. However, only half the matrix is referenced and modified.

rad_cchold_split_f

Compute a Cholesky decomposition of a Hermitian positive definite matrix.

Prototype

```
int rad_cchold_split_f(
                  rad_cchol_f *chold,
                  float       *A_re,
                  float       *A_im,
                  rad_stride   ldA);
```

Parameters

`chold`, structure, input.

`A_re`, real part of complex matrix, size n by n, modified in place.

`A_im`, imaginary part of complex matrix, size n by n, modified in place.

`ldA`, integer scalar, input.

Return Value

Error code.

Description

The Cholesky decomposition of a Hermitian positive definite n by n matrix A is given by $A = LL^{\mathrm{H}}$ where L is a lower triangular matrix.
There is not a utility function for accessing the factors.
Returns zero on success. The routine will fail if a leading minor is not positive definite.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the decomposition is required.

Errors

The input and input/output objects must conform to the following:

1. All objects must be valid.
2. The matrix `A` and the Cholesky decomposition object `chold` must be conformant.

Notes

The matrix A is assumed to be Hermitian. This property is not checked. There is no reduced storage format for Hermitian matrices so the full matrix must be specified. However, only half the matrix is referenced and modified.

rad_chold_create_f

Creates a Cholesky decomposition object.

Prototype

```
rad_chol_f * rad_chold_create_f(
                  rad_mat_uplo  uplo,
                  unsigned int  n);
```

Parameters

uplo, enumerated type, input.

RAD_TR_LOW	lower triangle
RAD_TR_UPP	upper triangle

n, integer scalar, input.

Return Value

structure.

Description

Creates a Cholesky decomposition object. The Cholesky decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.
The Cholesky decomposition object is used to compute the Cholesky decomposition of a symmetric positive definite n by n matrix A.
The Cholesky decomposition of a symmetric positive definite n by n matrix A is given by $A = LL^{\mathrm{T}}$ where L is a lower triangular matrix.
There is not a utility function for accessing the factors.
NULL is returned if the create fails.

Restrictions
Errors

The input parameters must conform to the following:

1. n is positive.
2. uplo is valid.

Notes

rad_cchold_create_f

Creates a Cholesky decomposition object.

Prototype

```
rad_cchol_f * rad_cchold_create_f(
                  rad_mat_uplo  uplo,
                  unsigned int  n);
```

Parameters

uplo, enumerated type, input.

RAD_TR_LOW	lower triangle
RAD_TR_UPP	upper triangle

n, integer scalar, input.

Return Value

structure.

Description

Creates a Cholesky decomposition object. The Cholesky decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.
The Cholesky decomposition object is used to compute the Cholesky decomposition of a Hermitian positive definite n by n matrix A.
The Cholesky decomposition of a Hermitian positive definite n by n matrix A is given by $A = LL^{\mathrm{H}}$ where L is a lower triangular matrix.
There is not a utility function for accessing the factors.
NULL is returned if the create fails.

Restrictions
Errors

The input parameters must conform to the following:

1. n is positive.
2. uplo is valid.

Notes

rad_Dchold_destroy_P

Destroy a Cholesky decomposition object.

Prototype

```
int rad_Dchold_destroy_P(
                    rad_Dchol_P *chold);
```

The following instances are supported:

`rad_chold_destroy_f`

`rad_cchold_destroy_f`

Parameters

`chold`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) a Cholesky decomposition object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The input object must conform to the following:

1. The Cholesky decomposition object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_Dchold_getattr_P

Returns the attributes of a Cholesky decomposition object.

Prototype

```
void rad_Dchold_getattr_P(
                  rad_Dchol_P      *chold,
                  rad_Dchol_attr_P *attr);
```

The following instances are supported:

`rad_chold_getattr_f`

`rad_cchold_getattr_f`

Parameters

`chold`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`rad_mat_uplo`	`uplo`	upper or lower triangular matrix
`unsigned int`	`n`	number of rows and columns in matrix

Return Value

none.

Description

Returns the attributes of a Cholesky decomposition object.

Restrictions
Errors

The input and output arguments must conform to the following:

1. The Cholesky decomposition object `chold` must be valid.
2. The attribute pointer `attr` must not be `NULL`.

Notes

rad_cholsol_f

Solve a symmetric positive definite linear system.

Prototype

```
int rad_cholsol_f(
                  rad_cchol_f   *chold,
                  float         *XB,
                  rad_stride     ldXB,
                  unsigned int   p);
```

Parameters

chold, structure, input. A Cholesky decomposition object for the matrix A.

XB, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solve the linear system $AX = B$ for a symmetric positive definite matrix A using the decomposition computed by rad_chold_f.
Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after rad_chold_f fails. This will result in a non-zero unsuccessful return value.

rad_ccholsol_f

Solve a Hermitian positive definite linear system.

Prototype

```
int rad_ccholsol_f(
                   rad_cchol_f   *chold,
                   void          *XB,
                   rad_stride     ldXB,
                   unsigned int   p);
```

Parameters

chold, structure, input. A Cholesky decomposition object for the matrix A.

XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solve the linear system $AX = B$ for a Hermitian positive definite matrix A using the decomposition computed by rad_cchold_f.
Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after rad_cchold_f fails. This will result in a non-zero unsuccessful return value.

rad_ccholsol_split_f

Solve a Hermitian positive definite linear system.

Prototype

```
int rad_ccholsol_split_f(
                  rad_cchol_f   *chold,
                  float         *XB_re,
                  float         *XB_im,
                  rad_stride     ldXB,
                  unsigned int   p);
```

Parameters

chold, structure, input. A Cholesky decomposition object for the matrix A.

XB_re, real part of complex matrix, size n by p, modified in place.

XB_im, imaginary part of complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solve the linear system $AX = B$ for a Hermitian positive definite matrix A using the decomposition computed by rad_cchold_f.
Returns zero on success, non-zero on failure.

Restrictions

Errors

Notes

It is safe to call this function after rad_cchold_f fails. This will result in a non-zero unsuccessful return value.

8.5 Overdetermined Linear System Solver

rad_qrd_f

rad_cqrd_f

rad_cqrd_split_f

rad_qrd_create_f

rad_cqrd_create_f

rad_Dqrd_destroy_P

rad_Dqrd_getattr_P

rad_qrdprodq_f

rad_cqrdprodq_f

rad_cqrdprodq_split_f

rad_qrdsolr_f

rad_cqrdsolr_f

rad_cqrdsolr_split_f

rad_qrsol_f

rad_cqrsol_f

rad_cqrsol_split_f

rad_qrd_f

Compute a QR decomposition of a matrix .

Prototype

```
int rad_qrd_f(
                    rad_cqr_f   *qrd,
                    float       *A,
                    rad_stride   ldA);
```

Parameters

qrd, structure, input.

A, real matrix, size m by n, modified in place.

ldA, integer scalar, input.

Return Value

Error code.

Description

Computes the QR decomposition of an m by n matrix A where $n \leqslant m$.
The QR decomposition of A is given by $A = QR$ where Q is an m by n orthogonal matrix ($Q^{\mathrm{T}}Q = I$) and R is an n by n upper triangular matrix. If A has full rank then R is non-singular.
The routine does not perform any column interchanges.
Returns zero on success. It will fail if A does not have full column rank.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero diagonal element of R was encountered.

rad_cqrd_f

Compute a QR decomposition of a matrix .

Prototype

```
int rad_cqrd_f(
                    rad_cqr_f   *qrd,
                    void        *A,
                    rad_stride   ldA);
```

Parameters

qrd, structure, input.

A, complex matrix, size m by n, modified in place.

ldA, integer scalar, input.

Return Value

Error code.

Description

Computes the QR decomposition of an m by n matrix A where $n \leqslant m$.
The QR decomposition of A is given by $A = QR$ where Q is an m by n unitary matrix ($Q^{\mathrm{H}}Q = I$) and R is an n by n upper triangular matrix. If A has full rank then R is non-singular. The routine does not perform any column interchanges.
Returns zero on success. It will fail if A does not have full column rank.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero diagonal element of R was encountered.

rad_cqrd_split_f

Compute a QR decomposition of a matrix .

Prototype

```
int rad_cqrd_split_f(
                    rad_cqr_f   *qrd,
                    float       *A_re,
                    float       *A_im,
                    rad_stride   ldA);
```

Parameters

qrd, structure, input.

A_re, real part of complex matrix, size m by n, modified in place.

A_im, imaginary part of complex matrix, size m by n, modified in place.

ldA, integer scalar, input.

Return Value

Error code.

Description

Computes the QR decomposition of an m by n matrix A where $n \leqslant m$.
The QR decomposition of A is given by $A = QR$ where Q is an m by n unitary matrix ($Q^{\mathrm{H}}Q = I$) and R is an n by n upper triangular matrix. If A has full rank then R is non-singular. The routine does not perform any column interchanges.
Returns zero on success. It will fail if A does not have full column rank.

Restrictions

The matrix A is overwritten by the decomposition and must not be modified as long as the factorisation is required.

Errors

Notes

The matrix A is assumed to be of full rank. This property is not checked. A positive return value indicates that an error occurred and a zero diagonal element of R was encountered.

rad_qrd_create_f

Create a QR decomposition object.

Prototype

```
rad_qr_f * rad_qrd_create_f(
                  unsigned int  m,
                  unsigned int  n,
                  rad_qrd_qopt  qopt);
```

Parameters

m, integer scalar, input.

n, integer scalar, input.

qopt, enumerated type, input.

RAD_QRD_NOSAVEQ	do not save Q
RAD_QRD_SAVEQ	save full Q
RAD_QRD_SAVEQ1	save skinny Q

Return Value

structure.

Description

Creates a QR decomposition object. The QR decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.
The QR decomposition of A is given by $A = QR$ where Q is an m by n orthogonal matrix ($Q^{\mathrm{T}}Q = I$) and R is an n by n upper triangular matrix, and $n \leqslant m$. If A has full rank then R is non-singular.
The matrix R will be generated and retained for later use. There is a flag to indicate whether Q is retained and, if so, in what format.
NULL is returned if the create fails.

Restrictions
Errors

The input arguments must conform to the following:

1. m and n positive with n $\leqslant$ m
2. qopt is valid.

Notes

rad_cqrd_create_f

Create a QR decomposition object.

Prototype

```
rad_cqr_f * rad_cqrd_create_f(
                  unsigned int  m,
                  unsigned int  n,
                  rad_qrd_qopt  qopt);
```

Parameters

m, integer scalar, input.

n, integer scalar, input.

qopt, enumerated type, input.

RAD_QRD_NOSAVEQ	do not save Q
RAD_QRD_SAVEQ	save full Q
RAD_QRD_SAVEQ1	save skinny Q

Return Value

structure.

Description

Creates a QR decomposition object. The QR decomposition object encapsulates the information concerning the properties of the decomposition and required workspace.
The QR decomposition of A is given by $A = QR$ where Q is an m by n unitary matrix ($Q^{\mathrm{H}}Q = I$) and R is an n by n upper triangular matrix, and $n \leqslant m$. If A has full rank then R is non-singular.
The matrix R will be generated and retained for later use. There is a flag to indicate whether Q is retained and, if so, in what format.
NULL is returned if the create fails.

Restrictions
Errors

The input arguments must conform to the following:

1. m and n positive with n $\leqslant$ m
2. qopt is valid.

Notes

rad_Dqrd_destroy_P

Destroy a QR decomposition object.

Prototype

```
int rad_Dqrd_destroy_P(
                    rad_Dqr_P *qrd);
```

The following instances are supported:

`rad_qrd_destroy_f`

`rad_cqrd_destroy_f`

Parameters

`qrd`, structure, input.

Return Value

Error code.

Description

Destroys (frees the memory used by) a QR decomposition object. Returns zero on success, non-zero on failure.

Restrictions
Errors

The input object must conform to the following:

1. The QR decomposition object must be valid. An argument of `NULL` is not an error.

Notes

An argument of `NULL` is not an error.

rad_Dqrd_getattr_P

Returns the attributes of a QR decomposition object.

Prototype

```
void rad_Dqrd_getattr_P(
                   rad_Dqr_P      *qrd,
                   rad_Dqr_attr_P *attr);
```

The following instances are supported:

`rad_qrd_getattr_f`

`rad_cqrd_getattr_f`

Parameters

`qrd`, structure, input.

`attr`, pointer to structure, output.
The attribute structure contains the following information:

`unsigned int`	`m`	number of rows in input matrix
`unsigned int`	`n`	number of columns in input matrix
`rad_qrd_qopt`	`Qopt`	matrix Q is saved/not saved

Return Value

none.

Description

Returns the attributes of a QR decomposition object.

Restrictions

Errors

Notes

rad_qrdprodq_f

Multiply a matrix by the matrix Q from a QR decomposition.

Prototype

```
int rad_qrdprodq_f(
                   rad_qr_f      *qrd,
                   rad_mat_op     opQ,
                   rad_mat_side   apQ,
                   float         *C,
                   rad_stride     ldC,
                   unsigned int   p);
```

Parameters

qrd, structure, input.

opQ, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose

apQ, enumerated type, input.

RAD_MAT_LSIDE	left side
RAD_MAT_RSIDE	right side

C, real matrix, size p by q, modified in place.

ldC, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

This function overwrites a matrix C with the product $\text{op}(Q) \cdot C$ if multiplied on the left, or $C \cdot \text{op}(Q)$ if multiplied on the right. The matrix Q is computed by rad_qrd_f (its size depends on which option was used to store it), and op() is either the identity or transpose operation.
In some cases, the output matrix is larger than C.
Returns zero on success, non-zero on failure.

Restrictions

Since the output data space may be larger than the input data space, it is required that the input allows storage in the block for the output data. This means the row stride and column stride must be calculated to accommodate the larger data space, whether it be input or output.

Errors

The arguments must conform to the following:

1. opQ is valid
2. apQ is valid
3. The matrix C and the QR decomposition object qrd must be conformant.
4. The QR decomposition object must have specified retaining the Q matrix when it was created.

Notes

It is safe to call this function after `rad_qrd_f`. fails. This will result in a non-zero unsuccessful return value.

rad_cqrdprodq_f

Multiply a matrix by the matrix Q from a QR decomposition.

Prototype

```
int rad_cqrdprodq_f(
                    rad_cqr_f     *qrd,
                    rad_mat_op     opQ,
                    rad_mat_side   apQ,
                    void          *C,
                    rad_stride     ldC,
                    unsigned int   p);
```

Parameters

`qrd`, structure, input.

`opQ`, enumerated type, input.

`RAD_MAT_NTRANS`	no transformation
`RAD_MAT_TRANS`	transpose
`RAD_MAT_HERM`	Hermitian (conjugate transpose)

`apQ`, enumerated type, input.

`RAD_MAT_LSIDE`	left side
`RAD_MAT_RSIDE`	right side

`C`, complex matrix, size p by q, modified in place.

`ldC`, integer scalar, input.

`p`, integer scalar, input.

Return Value

Error code.

Description

This function overwrites a matrix C with the product $\text{op}(Q) \cdot C$ if multiplied on the left, or $C \cdot \text{op}(Q)$ if multiplied on the right. The matrix Q is computed by `rad_cqrd_f` (its size depends on which option was used to store it), and op() is either the identity or (identity, transpose, or Hermitian). operation.
In some cases, the output matrix is larger than C.
Returns zero on success, non-zero on failure.

Restrictions

Since the output data space may be larger than the input data space, it is required that the input allows storage in the block for the output data. This means the row stride and column stride must be calculated to accommodate the larger data space, whether it be input or output.

Errors

The arguments must conform to the following:

1. `opQ` is valid
2. `apQ` is valid
3. The matrix `C` and the QR decomposition object `qrd` must be conformant.
4. The QR decomposition object must have specified retaining the Q matrix when it was created.

Notes

It is safe to call this function after `rad_cqrd_f`. fails. This will result in a non-zero unsuccessful return value.

rad_cqrdprodq_split_f

Multiply a matrix by the matrix Q from a QR decomposition.

Prototype

```
int rad_cqrdprodq_split_f(
                  rad_cqr_f     *qrd_re,
                  rad_cqr_f     *qrd_im,
                  rad_mat_op     opQ,
                  rad_mat_side   apQ,
                  float         *C_re,
                  float         *C_im,
                  rad_stride     ldC,
                  unsigned int   p);
```

Parameters

qrd_re, real part of structure, input.

qrd_im, imaginary part of structure, input.

opQ, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)

apQ, enumerated type, input.

RAD_MAT_LSIDE	left side
RAD_MAT_RSIDE	right side

C_re, real part of complex matrix, size p by q, modified in place.

C_im, imaginary part of complex matrix, size p by q, modified in place.

ldC, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

This function overwrites a matrix C with the product op$(Q) \cdot C$ if multiplied on the left, or $C \cdot$op(Q) if multiplied on the right. The matrix Q is computed by rad_cqrd_f (its size depends on which option was used to store it), and op() is either the identity or (identity, transpose, or Hermitian). operation.
In some cases, the output matrix is larger than C.
Returns zero on success, non-zero on failure.

Restrictions

Since the output data space may be larger than the input data space, it is required that the input allows storage in the block for the output data. This means the row stride and column stride must be calculated to accommodate the larger data space, whether it be input or output.

Errors

The arguments must conform to the following:

1. opQ is valid
2. apQ is valid
3. The matrix C and the QR decomposition object qrd must be conformant.
4. The QR decomposition object must have specified retaining the Q matrix when it was created.

Notes

It is safe to call this function after rad_cqrd_f. fails. This will result in a non-zero unsuccessful return value.

rad_qrdsolr_f

Solve linear system based on the matrix R, from QR decomposition of the matrix A.

Prototype

```
int rad_qrdsolr_f(
                  rad_qr_f      *qrd,
                  rad_mat_op     OpR,
                  float          alpha,
                  float         *XB,
                  rad_stride     ldXB,
                  unsigned int   p);
```

Parameters

qrd, structure, input. A QR decomposition object for the matrix A.

OpR, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose

alpha, real scalar, input.

XB, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solves a triangular linear system of the form op$(R)X = \alpha B$ where op() is either the identity or transpose operation, using the decomposition computed by rad_qrd_f.
R is an n by n upper triangular matrix; X and B are n by p matrices.
Returns zero on success, non-zero on failure.

Restrictions
Errors

The arguments must conform to the following:

1. All of the objects must be valid.
2. OpR is valid.
3. The matrix XB and the QR decomposition object qrd must be conformant.

Notes

It is safe to call this function after rad_qrd_f. fails. This will result in a non-zero unsuccessful return value.

rad_cqrdsolr_f

Solve linear system based on the matrix R, from QR decomposition of the matrix A.

Prototype

```
int rad_cqrdsolr_f(
                    rad_cqr_f     *qrd,
                    rad_mat_op     OpR,
                    void          *alpha,
                    void          *XB,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

qrd, structure, input. A QR decomposition object for the matrix A.

OpR, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)

alpha, complex scalar, input.

XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solves a triangular linear system of the form op$(R)X = \alpha B$ where op() is either the identity or (identity, transpose, or Hermitian). operation, using the decomposition computed by rad_cqrd_f.
R is an n by n upper triangular matrix; X and B are n by p matrices.
Returns zero on success, non-zero on failure.

Restrictions
Errors

The arguments must conform to the following:

1. All of the objects must be valid.
2. OpR is valid.
3. The matrix XB and the QR decomposition object qrd must be conformant.

Notes

It is safe to call this function after rad_cqrd_f. fails. This will result in a non-zero unsuccessful return value.

rad_cqrdsolr_split_f

Solve linear system based on the matrix R, from QR decomposition of the matrix A.

Prototype

```
int rad_cqrdsolr_split_f(
                    rad_cqr_f     *qrd_re,
                    rad_cqr_f     *qrd_im,
                    rad_mat_op     OpR,
                    float         *alpha_re,
                    float         *alpha_im,
                    float         *XB_re,
                    float         *XB_im,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

qrd_re, real part of structure, input.

qrd_im, imaginary part of structure, input. A QR decomposition object for the matrix A.

OpR, enumerated type, input.

RAD_MAT_NTRANS	no transformation
RAD_MAT_TRANS	transpose
RAD_MAT_HERM	Hermitian (conjugate transpose)

alpha_re, real part of complex scalar, input.

alpha_im, imaginary part of complex scalar, input.

XB_re, real part of complex matrix, size n by p, modified in place.

XB_im, imaginary part of complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Solves a triangular linear system of the form $\mathrm{op}(R)X = \alpha B$ where op() is either the identity or (identity, transpose, or Hermitian). operation, using the decomposition computed by rad_cqrd_f.
R is an n by n upper triangular matrix; X and B are n by p matrices.
Returns zero on success, non-zero on failure.

Restrictions
Errors

The arguments must conform to the following:

1. All of the objects must be valid.
2. OpR is valid.
3. The matrix XB and the QR decomposition object qrd must be conformant.

Notes

It is safe to call this function after `rad_cqrd_f`. fails. This will result in a non-zero unsuccessful return value.

rad_qrsol_f

Solve either a linear covariance or linear least squares problem.

Prototype

```
int rad_qrsol_f(
                    rad_qr_f      *qrd,
                    rad_qrd_prob   prob,
                    float         *XB,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

qrd, structure, input. A QR decomposition object for the matrix A.

prob, enumerated type, input.

RAD_COV	solve a covariance linear system problem
RAD_LLS	solve a linear least squares problem

XB, real matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Let A be an m by n matrix of rank n with $n \leqslant m$, and let B be a matrix of size n by p for a covariance problem or m by p for a least squares problem.
This routine solves one of the following problems using the decomposition computed by rad_qrd_f

a covariance linear system problem $A^{\mathrm{T}}AX = B$

a linear least squares problem $\min|AX - B|_2$.

Restrictions

This routine will fail if A does not have full rank.

Errors

The arguments must conform to the following:

1. The matrix XB and the QR decomposition object qrd must be conformant.
2. prob is valid.

Notes

It is safe to call this function after rad_qrd_f fails. This will result in a non-zero unsuccessful return value.

rad_cqrsol_f

Solve either a linear covariance or linear least squares problem.

Prototype

```
int rad_cqrsol_f(
                    rad_cqr_f     *qrd,
                    rad_qrd_prob   prob,
                    void          *XB,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

qrd, structure, input. A QR decomposition object for the matrix A.

prob, enumerated type, input.

RAD_COV	solve a covariance linear system problem
RAD_LLS	solve a linear least squares problem

XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Let A be an m by n matrix of rank n with $n \leqslant m$, and let B be a matrix of size n by p for a covariance problem or m by p for a least squares problem.
This routine solves one of the following problems using the decomposition computed by rad_qrd_f

a covariance linear system problem $A^{\mathrm{H}}AX = B$

a linear least squares problem $\min|AX - B|_2$.

Restrictions

This routine will fail if A does not have full rank.

Errors

The arguments must conform to the following:

1. The matrix XB and the QR decomposition object qrd must be conformant.
2. prob is valid.

Notes

It is safe to call this function after rad_cqrd_f fails. This will result in a non-zero unsuccessful return value.

rad_cqrsol_split_f

Solve either a linear covariance or linear least squares problem.

Prototype

```
int rad_cqrsol_split_f(
                    rad_cqr_f     *qrd_re,
                    rad_cqr_f     *qrd_im,
                    rad_qrd_prob   prob,
                    void          *XB,
                    rad_stride     ldXB,
                    unsigned int   p);
```

Parameters

qrd_re, real part of structure, input.

qrd_im, imaginary part of structure, input. A QR decomposition object for the matrix A.

prob, enumerated type, input.

RAD_COV	solve a covariance linear system problem
RAD_LLS	solve a linear least squares problem

XB, complex matrix, size n by p, modified in place. On input, the matrix B. On output, the matrix X.

ldXB, integer scalar, input.

p, integer scalar, input.

Return Value

Error code.

Description

Let A be an m by n matrix of rank n with $n \leqslant m$, and let B be a matrix of size n by p for a covariance problem or m by p for a least squares problem.
This routine solves one of the following problems using the decomposition computed by rad_qrd_f

a covariance linear system problem $A^{\mathrm{H}}AX = B$

a linear least squares problem $\min|AX - B|_2$.

Restrictions

This routine will fail if A does not have full rank.

Errors

The arguments must conform to the following:

1. The matrix XB and the QR decomposition object qrd must be conformant.
2. prob is valid.

Notes

It is safe to call this function after rad_cqrd_f fails. This will result in a non-zero unsuccessful return value.