# Math Vector Library 1.1
# Reference Manual (C/C++)
# DD-00002-011

Jan Adelsbach

February 17, 2025

# Contents

# 1 About this Guide

## 1.1 Legal Information



## 1.2 Feedback and Contact

For feedback on this document, please use the following email address:
techpubs@adelsbach-research.eu

Please include the page number or a link to the page.

For general contact details, please visit https://adelsbach-research.eu/contact.

## 1.3 Introduction

This manual describes the *Application Programming Interface* (API) of the *Math Vector Library* for the C and C++ programming language families.

## 1.4 Audience for This Guide

The audience of this guide is assumed to be C or C++ programmers who understand the basic concepts of at least one of the aforementioned programming languages.

Familiarity with pointer based arrays in C/C++ is strongly recommended.

## 1.5 How to Use This Guide

This guide first describes some general programming details of the library and then documents each function individually.

The documentation for each function applies both the *single* and *double* precision versions. The former can be differentiated by a suffix letter `f`.

## 1.6 Conventions Used in This Guide

$x$
: Normal math typesetting represents a normal variable.

$\mathbf{x}$
: Bold math typesetting represents a vector.

`Mono`
: Monospace typesetting represents C function names, variables or data types.

# 2 Overview

## 2.1 Introduction

The *Math Vector Library* is a high-performance function library with vectorized versions of standard mathematical functions. The functions can operate both on dense and strided vectors, the latter can be supplied individually for result and operand vectors. Stride only executes the function on every $n$-th element leaving the elements in between untouched.

This manual describes the *Application Programming Interface* (API) of the mathematics vector functions.

## 2.2 Thread Safety

All routines in the library are completely thread-safe, as long as the data supplied in arguments is exclusive to the current thread.

## 2.3 SIMD/SPMD Unit Usage

This library makes excessive use of *Single Instruction Multiple Data* (SIMD) or *Single program Multiple Data* (SPMD) style extensions of the respective processor platform. It thereby abides by the standard system calling conventions when utilizing such.

## 2.4 Performance Characteristics

All subroutines in this library have a performance characteristic of $O(n)$. The routines may have different execution profiles depending upon the arguments supplied.

## 2.5 Extended Vector Sizes (`_64` suffix functions)

The default subroutines use 32bit integers for the array sizes, this limits the applicable size of any vector passed to $2^{31}-1$. Since for some applications this may be a limitation the Math Vector library also contains additional variants taking 64bit long integers, these then limit the applicable array size to $2^{63}-1$. These functions with 64bit integer sizes have `_64` suffixes to their function names but are otherwise functionally identical to the normal library functions, aside from using 64bit integers for all scalar arguments.

From a performance point of view the `_64` suffix versions will be slightly slower compared to the 32bit counterparts due to the long integer arithmetic involved. This is especially the case on processor architectures, where 64bit integer arithmetic is emulated using 32bit integer instructions.

# 3 Utility

## 3.1 `mvecver` - Version query

```
# include <mvec.h>

void mvecver(int *major, int *minor);
```

Queries the version of the library and stores the *major* and *minor* version numbers in the respective arguments.

### 3.1.1 Parameters

`MAJOR` - **INTEGER** *EXIT:* The major version number of the library.

`MINOR` - **INTEGER** *EXIT:* The minor version number of the library

# 4 Rounding

## 4.1 `vfloor` - Vector round down

```
# include <mvec.h>

void vfloor (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vfloorf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vfloor_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vfloorf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function rounds the elements of **x** to the nearest integral part less or equal than |**x**| and stores the result in **y**.

### 4.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 4.2 vceil - Vector round up

```
# include <mvec.h>

void vceil (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vceilf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vceil_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vceilf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function rounds the elements of **x** to the nearest integral part greater or equal than |**x**| and stores the result in **y**.

### 4.2.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 4.3 vtrunc - Vector truncate

```
# include <mvec.h>

void vtrunc (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vtruncf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vtrunc_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vtruncf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function truncates the elements of **x** to the nearest integral part lower or equal than |**x**| and stores the result in **y**.

### 4.3.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 4.4 vround - Vector rounding

```
# include <mvec.h>

void vround (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vroundf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vround_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vroundf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function rounds the elements of **x** to the nearest integral part and stores the result in **y**.

### 4.4.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 5 Roots

## 5.1 `vsqrt` - Vector square root $\sqrt{\mathbf{x}}$

```
# include <mvec.h>

void vsqrt (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vsqrtf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vsqrt_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vsqrtf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sqrt{\mathbf{x}}$$

### 5.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 5.2 vrsqrt - Vector reciprocal square root $1/\sqrt{\mathbf{x}}$

```
# include <mvec.h>

void vrsqrt (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vrsqrtf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vrsqrt_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vrsqrtf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \frac{1}{\sqrt{\mathbf{x}}}$$

### 5.2.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 5.3 vcbrt - Vector cube root $\sqrt[3]{\mathbf{x}}$

```
# include <mvec.h>

void vcbrt (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vcbrtf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vcbrt_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vcbrtf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sqrt[3]{\mathbf{x}}$$

### 5.3.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 5.4 vrcbrt - Vector reciprocal cube root $1/\sqrt[3]{\mathbf{x}}$

```
# include <mvec.h>

void vrcbrt (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vrcbrtf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vrcbrt_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vrcbrtf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \frac{1}{\sqrt[3]{\mathbf{x}}}$$

### 5.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 6 Trigonometric Functions

## 6.1 `vsin` - Vector sine sin(**x**)

```
# include <mvec.h>

void vsin (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vsinf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vsin_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vsinf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sin(\mathbf{x})$$

### 6.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.2 vcos - Vector cosine cos(**x**)

```
# include <mvec.h>

void vcos (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vcosf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vcos_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vcosf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cos(\mathbf{x})$$

### 6.2.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.3 vtan - Vector tangent tan(**x**)

```
# include <mvec.h>

void vtan (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vtanf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vtan_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vtanf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tan(\mathbf{x})$$

### 6.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.4 vasin - Vector arcsine $\sin^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vasin (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vasinf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vasin_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vasinf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sin^{-1}(\mathbf{x})$$

### 6.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.5 vacos - Vector arccosine $\cos^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vacos (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vacosf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vacos_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vacosf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cos^{-1}(\mathbf{x})$$

### 6.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.6 vatan - Vector arctangent $\tan^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vatan (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vatanf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vatan_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vatanf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tan^{-1}(\mathbf{x})$$

### 6.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.7 vatan2 - Vector arctangent $\tan^{-1}(\mathbf{x}/\mathbf{y})$

```
# include <mvec.h>

void vatan2 (int n, double *restrict z, int incz, const double *restrict x,
             int incx, const double *restrict y, int incy);
void vatan2f(int n, float *restrict z, int incz, const float *restrict x,
             int incx, const float *restrict y, int incy);

void vatan2_64 (long n, double *restrict z, long incz, const double *restrict x,
                long incx, const double *restrict y, long incy);
void vatan2f_64(long n, float *restrict z, long incz, const float *restrict x,
                long incx, const float *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes:

$$\mathbf{z} = \tan^{-1}(\mathbf{x}/\mathbf{y})$$

### 6.7.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.8 vsind - Vector sine sin(**x**) (degrees)

```
# include <mvec.h>

void vsind (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vsindf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vsind_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vsindf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** with angles in degrees (°) and a result vector **y**, this function computes:

$$\mathbf{y} = \sin(\mathbf{x})$$

### 6.8.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.9 vcosd - Vector cosine cos(**x**) (degrees)

```
# include <mvec.h>

void vcosd (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vcosdf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vcosd_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vcosdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** with angles in degrees (°) and a result vector **y**, this function computes:

$$\mathbf{y} = \cos(\mathbf{x})$$

### 6.9.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.10 vtand - Vector tangent tan(**x**) (degrees)

```
# include <mvec.h>

void vtand (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vtandf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vtand_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vtandf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** with angles in degrees (°) and a result vector **y**, this function computes:

$$\mathbf{y} = \tan(\mathbf{x})$$

### 6.10.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.11 vasind - Vector arcsine $\sin^{-1}(\mathbf{x})$ (degrees)

```
# include <mvec.h>

void vasind (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vasindf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vasind_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vasindf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sin^{-1}(\mathbf{x})$$

### 6.11.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.12 vacosd - Vector arccosine $\cos^{-1}(\mathbf{x})$ (degrees)

```
# include <mvec.h>

void vacosd (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vacosdf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vacosd_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vacosdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cos^{-1}(\mathbf{x})$$

### 6.12.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.13 vatand - Vector arctangent $\tan^{-1}(\mathbf{x})$ (degrees)

```
# include <mvec.h>

void vatand (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vatandf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vatand_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vatandf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tan^{-1}(\mathbf{x})$$

### 6.13.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.14 vsinpi - Vector half-cycle sine sin(π**x**)

```
# include <mvec.h>

void vsinpi (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vsinpif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vsinpi_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vsinpif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** with angles in degrees (°) and a result vector **y**, this function computes:

$$\mathbf{y} = \sin(\pi\mathbf{x})$$

### 6.14.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.15 vcospi - Vector half-cycle cosine cos(π**x**)

```
# include <mvec.h>

void vcospi (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vcospif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vcospi_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vcospif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** with angles in degrees (°) and a result vector **y**, this function computes:

$$\mathbf{y} = \cos(\pi\mathbf{x})$$

### 6.15.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.16 vtanpi - Vector half-cycle tangent tan(πx)

```
# include <mvec.h>

void vtanpi (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vtanpif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vtanpi_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vtanpif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** with angles in degrees (°) and a result vector **y**, this function computes:

$$\mathbf{y} = \tan(\pi\mathbf{x})$$

### 6.16.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.17 vasinpi - Vector half-cycle arcsine $\sin^{-1}(\mathbf{x})/\pi$

```
# include <mvec.h>

void vasinpi (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vasinpif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vasinpi_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vasinpif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sin^{-1}(\mathbf{x})/\pi$$

### 6.17.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.18 vacospi - Vector half-cycle arccosine $\cos^{-1}(\mathbf{x})/\pi$

```
# include <mvec.h>

void vacospi (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vacospif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vacospi_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vacospif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cos^{-1}(\mathbf{x})/\pi$$

### 6.18.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 6.19 vatanpi - Vector half-cycle arctangent $\tan^{-1}(\mathbf{x})/\pi$

```
# include <mvec.h>

void vatanpi (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vatanpif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vatanpi_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vatanpif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tan^{-1}(\mathbf{x})/\pi$$

### 6.19.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 7 Hypergeometric Functions

## 7.1 vsinh - Vector hypergeometric sine sinh(**x**)

```
# include <mvec.h>

void vsinh (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vsinhf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vsinh_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vsinhf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sinh(\mathbf{x})$$

### 7.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 7.2 vcosh - Vector hypergeometric cosine cosh(**x**)

```
# include <mvec.h>

void vcosh (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vcoshf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vcosh_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vcoshf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cosh(\mathbf{x})$$

### 7.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 7.3 vtanh - Vector hypergeometric tangent tanh(**x**)

```
# include <mvec.h>

void vtanh (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vtanhf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vtanh_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vtanhf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tanh(\mathbf{x})$$

### 7.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 7.4 vasinh - Vector hypergeometric arcsine $\sinh^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vasinh (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vasinhf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vasinh_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vasinhf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sinh^{-1}(\mathbf{x})$$

### 7.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 7.5 vacosh - Vector hypergeometric arccosine $\cosh^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vacosh (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vacoshf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vacosh_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vacoshf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cosh^{-1}(\mathbf{x})$$

### 7.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 7.6 vatanh - Vector hypergeometric arctangent $\tanh^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vatanh (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vatanhf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vatanh_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vatanhf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tanh^{-1}(\mathbf{x})$$

### 7.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 8 Exponentials and Logarithms

## 8.1 `vexp` - Vector exponential $e^{\mathbf{x}}$

```
# include <mvec.h>

void vexp (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vexpf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vexp_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vexpf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = e^{\mathbf{x}}$$

### 8.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.2 vexpm1 - Vector exponential $e^{\mathbf{x}}-1$

```
# include <mvec.h>

void vexpm1 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vexpm1f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vexpm1_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vexpm1f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = e^{\mathbf{x}} - 1$$

### 8.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.3 vexp2 - Vector binary exponential $2^{\mathbf{x}}$

```
# include <mvec.h>

void vexp2 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vexp2f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vexp2_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vexp2f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = 2^{\mathbf{x}}$$

### 8.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.4 vexp2m1 - Vector binary exponential minus one $2^{\mathbf{x}}-1$

```
# include <mvec.h>

void vexp2m1 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vexp2m1f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vexp2m1_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vexp2m1f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = 2^{\mathbf{x}} - 1$$

### 8.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.5 vexp10 - Vector natural exponential $10^{\mathbf{x}}$

```
# include <mvec.h>

void vexp10 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vexp10f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vexp10_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vexp10f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = 10^{\mathbf{x}}$$

### 8.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.6 vexp10m1 - Vector natural exponential minus one $10^{\mathbf{x}}-1$

```
# include <mvec.h>

void vexp10m1 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vexp10m1f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vexp10m1_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vexp10m1f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = 10^{\mathbf{x}} - 1$$

### 8.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.7 vlog - Vector logarithm log(**x**)

```
# include <mvec.h>

void vlog (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vlogf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vlog_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vlogf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \log(\mathbf{x})$$

### 8.7.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.8 vlog2 - Vector binary logarithm $\log_2(\mathbf{x})$

```
# include <mvec.h>

void vlog2 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vlog2f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vlog2_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vlog2f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \log_2(\mathbf{x})$$

### 8.8.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.9 vlog10 - Vector base-10 logarithm $\log_{10}(\mathbf{x})$

```
# include <mvec.h>

void vlog10 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vlog10f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vlog10_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vlog10f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \log_{10}(\mathbf{x})$$

### 8.9.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.10 vlog1p - Vector logarithm log(**x**+1)

```
# include <mvec.h>

void vlog1p (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vlog1pf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vlog1p_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vlog1pf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \log(\mathbf{x}+1)$$

### 8.10.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.11 vpow - Vector power $\mathbf{x}^{\mathbf{y}}$

```
# include <mvec.h>

void vpow (int n, double *restrict z, int incz, const double *restrict x,
           int incx, const double *restrict y, int incy);
void vpowf(int n, float *restrict z, int incz, const float *restrict x,
           int incx, const float *restrict y, int incy);

void vpow_64 (long n, double *restrict z, long incz, const double *restrict x,
              long incx, const double *restrict y, long incy);
void vpowf_64(long n, float *restrict z, long incz, const float *restrict x,
              long incx, const float *restrict y, long incy);
```

Given an input vectors **x** and **y** as well as a result vector **z**, this function computes:

$$\mathbf{z} = \mathbf{x}^{\mathbf{y}}$$

### 8.11.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 8.12 vpows - Vector power scalar exponent$\mathbf{x}^y$

```
# include <mvec.h>

void vpows (int n, double *restrict z, int incz, const double *restrict x,
            int incx, double y);
void vpowsf(int n, float *restrict z, int incz, const float *restrict x,
            int incx, float y);

void vpows_64 (long n, double *restrict z, long incz, const double *restrict x,
               long incx, double y);
void vpowsf_64(long n, float *restrict z, long incz, const float *restrict x,
               long incx, float y);
```

Given an input vectors **x** and a scalar value $y$ as well as a result vector **z**, this function computes:

$$\mathbf{z} = \mathbf{x}^y$$

### 8.12.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **REAL** *ENTRY:* Exponentiation value $y$.

## 8.13 vmod - Vector modulus mod(**x**,**y**)

```
# include <mvec.h>

void vmod (int n, double *restrict z, int incz, const double *restrict x,
           int incx, const double *restrict y, int incy);
void vmodf(int n, float *restrict z, int incz, const float *restrict x,
           int incx, const float *restrict y, int incy);

void vmod_64 (long n, double *restrict z, long incz, const double *restrict x,
              long incx, const double *restrict y, long incy);
void vmodf_64(long n, float *restrict z, long incz, const float *restrict x,
              long incx, const float *restrict y, long incy);
```

Given an input vectors **x** and **y** as well as a result vector **z**, this function computes:

$$\mathbf{z} = \text{mod}(\mathbf{x},\mathbf{y})$$

### 8.13.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 9 Error Functions

## 9.1 `verf` - Vector error function erf(**x**)

```
# include <mvec.h>

void verf (int n, double *restrict y, int incy, const double *restrict x, int incx);
void verff(int n, float *restrict y, int incy, const float *restrict x, int incx);

void verf_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void verff_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \mathrm{erf}(\mathbf{x})$$

### 9.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 9.2 verfinv - Vector inverse error function erf$^{-1}$(**x**)

```
# include <mvec.h>

void verfinv (int n, double *restrict y, int incy, const double *restrict x, int incx);
void verfinvf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void verfinv_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void verfinvf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \text{erf}^{-1}(\mathbf{x})$$

### 9.2.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 9.3 verfc - Vector complementary error function erfc(**x**)

```
# include <mvec.h>

void verfc (int n, double *restrict y, int incy, const double *restrict x, int incx);
void verfcf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void verfc_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void verfcf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \text{erfc}(\mathbf{x})$$

### 9.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 9.4 verfcinv - Vector inverse complementary error function erfc$^{-1}$(**x**)

```
# include <mvec.h>

void verfcinv (int n, double *restrict y, int incy, const double *restrict x, int incx);
void verfcinvf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void verfcinv_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void verfcinvf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \text{erfc}^{-1}(\mathbf{x})$$

### 9.4.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 10 Bessel Functions

## 10.1 vbesj0 - Vector Bessel Function $J_0(\mathbf{x})$

```
# include <mvec.h>

void vbesj0 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vbesj0f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vbesj0_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vbesj0f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = J_0(\mathbf{x})$$

### 10.1.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 10.2 vbesy0 - Vector Bessel Function $Y_0(\mathbf{x})$

```
# include <mvec.h>

void vbesy0 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vbesy0f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vbesy0_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vbesy0f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = Y_0(\mathbf{x})$$

### 10.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 10.3 vbesj1 - Vector Bessel Function $J_1(\mathbf{x})$

```
# include <mvec.h>

void vbesj1 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vbesj1f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vbesj1_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vbesj1f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = J_1(\mathbf{x})$$

### 10.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 10.4 vbesy1 - Vector Bessel Function $Y_1(\mathbf{x})$

```
# include <mvec.h>

void vbesy1 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vbesy1f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vbesy1_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vbesy1f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = Y_1(\mathbf{x})$$

### 10.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 10.5 vbesjn - Vector Bessel Function $J_n(\mathbf{x})$

```
# include <mvec.h>

void vbesjn (int n, double *restrict y, int incy, int k, const double *restrict x, int incx);
void vbesjnf(int n, float *restrict y, int incy, int k, const float *restrict x, int incx);

void vbesjn_64 (long n, double *restrict y, long incy, long k, const double *restrict x, long incx);
void vbesjnf_64(long n, float *restrict y, long incy, long k, const float *restrict x, long incx);
```

Given an input vector $\mathbf{x}$, an order $k$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = J_k(\mathbf{x})$$

### 10.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

K - **INTEGER** *ENTRY:* Order $k$.

X - **ARRAY OF REAL** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 10.6 vbesyn - Vector Bessel Function $Y_n(\mathbf{x})$

```
# include <mvec.h>

void vbesyn (int n, double *restrict y, int incy, int k, const double *restrict x, int incx);
void vbesynf(int n, float *restrict y, int incy, int k, const float *restrict x, int incx);

void vbesyn_64 (long n, double *restrict y, long incy, long k, const double *restrict x, long incx);
void vbesynf_64(long n, float *restrict y, long incy, long k, const float *restrict x, long incx);
```

Given an input vector $\mathbf{x}$, an order $k$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = Y_k(\mathbf{x})$$

### 10.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

K - **INTEGER** *ENTRY:* Order $k$.

X - **ARRAY OF REAL** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 11 Gamma Function Related

## 11.1 `vlgamma` - Vector Log-Gamma $\log\Gamma(\mathbf{x})$

```
# include <mvec.h>

void vlgamma (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vlgammaf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vlgamma_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vlgammaf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \log\Gamma(\mathbf{x})$$

### 11.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 12 Einstein Functions

## 12.1 veinst1 - Vector Einstein function $E_1(\mathbf{x})$

```
# include <mvec.h>

void veinst1 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void veinst1f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void veinst1_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void veinst1f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = E_1(\mathbf{x}) = \frac{\mathbf{x}^2 e^{\mathbf{x}}}{(e^{\mathbf{x}} - 1)^2}$$

### 12.1.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 12.2 veinst2 - Vector Einstein function $E_2(\mathbf{x})$

```
# include <mvec.h>

void veinst2 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void veinst2f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void veinst2_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void veinst2f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = E_2(\mathbf{x}) = \frac{\mathbf{x}}{e^{\mathbf{x}} - 1}$$

### 12.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 12.3 veinst3 - Vector Einstein function $E_3(\mathbf{x})$

```
# include <mvec.h>

void veinst3 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void veinst3f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void veinst3_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void veinst3f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = E_3(\mathbf{x}) = \log(1 - e^{-\mathbf{x}})$$

### 12.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 12.4 veinst4 - Vector Einstein function $E_4(\mathbf{x})$

```
# include <mvec.h>

void veinst4 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void veinst4f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void veinst4_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void veinst4f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = E_4(\mathbf{x}) = \frac{\mathbf{x}}{e^{\mathbf{x}} - 1} - \log(1 - e^{-\mathbf{x}})$$

### 12.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 13 Integrals

## 13.1 vsi - Vector Sine Integral Si(x)

```
# include <mvec.h>

void vsi (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vsif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vsi_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vsif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \mathrm{Si}(\mathbf{x}) = \int_0^{\mathbf{x}} \frac{\sin(\mathbf{x})}{\mathbf{x}}\, \mathrm{dx}$$

### 13.1.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 13.2 vci - Vector Cosine Integral Ci(**x**)

```
# include <mvec.h>

void vci (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vcif(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vci_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vcif_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \mathrm{Ci}(\mathbf{x}) = \int_0^{\mathbf{x}} \frac{\cos(\mathbf{x})}{\mathbf{x}}\, \mathrm{dx}$$

### 13.2.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 13.3 vti2 - Vector Inverse Tangent Integral Ti2(**x**)

```
# include <mvec.h>

void vti2 (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vti2f(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vti2_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vti2f_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \mathrm{Ti2}(\mathbf{x}) = \int_0^{\mathbf{x}} \frac{\tan^{-1}(\mathbf{x})}{\mathbf{x}}\, \mathrm{dx}$$

### 13.3.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 14 Other Functions

## 14.1 vabs - Vector absolute value |x|

```
# include <mvec.h>

void vabs (int n, double *restrict y, int incy, const double *restrict x, int incx);
void vabsf(int n, float *restrict y, int incy, const float *restrict x, int incx);

void vabs_64 (long n, double *restrict y, long incy, const double *restrict x, long incx);
void vabsf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = |\mathbf{x}|$$

### 14.1.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 14.2 vhypot - Vector euclidean distance $\sqrt{\mathbf{x}^2+\mathbf{y}^2}$

```
# include <mvec.h>

void vhypot (int n, double *restrict z, int incz, const double *restrict x,
             int incx, const double *restrict y, int incy);
void vhypotf(int n, float *restrict z, int incz, const float *restrict x,
             int incx, const float *restrict y, int incy);

void vhypot_64 (long n, double *restrict z, long incz, const double *restrict x,
                long incx, const double *restrict y, long incy);
void vhypotf_64(long n, float *restrict z, long incz, const float *restrict x,
                long incx, const float *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes:

$$\mathbf{z} = \sqrt{\mathbf{x}^2+\mathbf{y}^2}$$

### 14.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 14.3 vrem - Vector remainder

```
# include <mvec.h>

void vrem (int n, double *restrict z, int incz, const double *restrict x,
           int incx, const double *restrict y, int incy);
void vremf(int n, float *restrict z, int incz, const float *restrict x,
           int incx, const float *restrict y, int incy);

void vrem_64 (long n, double *restrict z, long incz, const double *restrict x,
              long incx, const double *restrict y, long incy);
void vremf_64(long n, float *restrict z, long incz, const float *restrict x,
              long incx, const float *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes the remainder of dividing $x$ by $y$ and stores the result in $z$.

### 14.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 15 Arithmetic Functions

## 15.1 vadd - Vector addition $\mathbf{x}+\mathbf{y}$

```
# include <mvec.h>

void vadd (int n, double *restrict z, int incz, const double *restrict x,
           int incx, const double *restrict y, int incy);
void vaddf(int n, float *restrict z, int incz, const float *restrict x,
           int incx, const float *restrict y, int incy);

void vadd_64 (long n, double *restrict z, long incz, const double *restrict x,
              long incx, const double *restrict y, long incy);
void vaddf_64(long n, float *restrict z, long incz, const float *restrict x,
              long incx, const float *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{x} + \mathbf{y}$$

### 15.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

`Z` - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

`INCZ` - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`Y` - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 15.2 vsadd - Vector scalar addition $\mathbf{x}+\alpha$

```
# include <mvec.h>

void vsadd (int n, double *restrict y, int incy, const double *restrict x, int incx, double a);
void vsaddf(int n, float *restrict y, int incy, const float *restrict x, int incx, float a);

void vsadd_64 (long n, double *restrict y, long incy, double *restrict x, long incx, double a);
void vsaddf_64(long n, float *restrict y, long incy, float *restrict x, long incx, float a);
```

Given an input vector $\mathbf{x}$ and a scalar constant $\alpha$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \mathbf{x} + \alpha$$

### 15.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **REAL** *ENTRY:* Scalar constant $\alpha$.

## 15.3 vsub - Vector subtraction x − y

```
# include <mvec.h>

void vsub (int n, double *restrict z, int incz, const double *restrict x,
           int incx, const double *restrict y, int incy);
void vsubf(int n, float *restrict z, int incz, const float *restrict x,
           int incx, const float *restrict y, int incy);

void vsub_64 (long n, double *restrict z, long incz, const double *restrict x,
              long incx, const double *restrict y, long incy);
void vsubf_64(long n, float *restrict z, long incz, const float *restrict x,
              long incx, const float *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{x} - \mathbf{y}$$

### 15.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 15.4 vssub - Vector scalar subtraction $\mathbf{x} - \alpha$

```
# include <mvec.h>

void vssub (int n, double *restrict y, int incy, const double *restrict x, int incx, double a);
void vssubf(int n, float *restrict y, int incy, const float *restrict x, int incx, float a);

void vssub_64 (long n, double *restrict y, long incy, double *restrict x, long incx, double a);
void vssubf_64(long n, float *restrict y, long incy, float *restrict x, long incx, float a);
```

Given an input vector $\mathbf{x}$ and a scalar constant $\alpha$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \mathbf{x} - \alpha$$

### 15.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **REAL** *ENTRY:* Scalar constant $\alpha$.

## 15.5 vmul - Vector multiplication xy

```
# include <mvec.h>

void vmul (int n, double *restrict z, int incz, const double *restrict x,
           int incx, const double *restrict y, int incy);
void vmulf(int n, float *restrict z, int incz, const float *restrict x,
           int incx, const float *restrict y, int incy);

void vmul_64 (long n, double *restrict z, long incz, const double *restrict x,
              long incx, const double *restrict y, long incy);
void vmulf_64(long n, float *restrict z, long incz, const float *restrict x,
              long incx, const float *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{xy}$$

### 15.5.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

`Z` - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

`INCZ` - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`Y` - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 15.6 vsmul - Vector scalar multiplication $\alpha\mathbf{x}$

```
# include <mvec.h>

void vsmul (int n, double *restrict y, int incy, const double *restrict x, int incx, double a);
void vsmulf(int n, float *restrict y, int incy, const float *restrict x, int incx, float a);

void vsmul_64 (long n, double *restrict y, long incy, double *restrict x, long incx, double a);
void vsmulf_64(long n, float *restrict y, long incy, float *restrict x, long incx, float a);
```

Given an input vector **x** and a scalar constant $\alpha$ and a result vector **y**, this function computes:

$$\mathbf{y} = \alpha\mathbf{x}$$

### 15.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **REAL** *ENTRY:* Scalar constant $\alpha$.

## 15.7 vdiv - Vector division x/y

```
# include <mvec.h>

void vdiv (int n, double *restrict z, int incz, const double *restrict x,
           int incx, const double *restrict y, int incy);
void vdivf(int n, float *restrict z, int incz, const float *restrict x,
           int incx, const float *restrict y, int incy);

void vdiv_64 (long n, double *restrict z, long incz, const double *restrict x,
              long incx, const double *restrict y, long incy);
void vdivf_64(long n, float *restrict z, long incz, const float *restrict x,
              long incx, const float *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{x}/\mathbf{y}$$

### 15.7.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF REAL** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF REAL** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 15.8 vsdiv - Vector scalar division **x**/$\alpha$

```
# include <mvec.h>

void vsdiv (int n, double *restrict y, int incy, const double *restrict x, int incx, double a);
void vsdivf(int n, float *restrict y, int incy, const float *restrict x, int incx, float a);

void vsdiv_64 (long n, double *restrict y, long incy, double *restrict x, long incx, double a);
void vsdivf_64(long n, float *restrict y, long incy, float *restrict x, long incx, float a);
```

Given an input vector **x** and a scalar constant $\alpha$ and a result vector **y**, this function computes:

$$\mathbf{y} = \mathbf{x}/\alpha$$

### 15.8.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **REAL** *ENTRY:* Scalar constant $\alpha$.
*CONSTRAINT:* $\alpha \neq 0$.

## 15.9 vrecp - Vector reciprocal 1/**x**

```
# include <mvec.h>

void vrecp (int n, double *restrict z, int incz, const double *restrict x,
            int incx, const double *restrict y, int incy);
void vrecpf(int n, float *restrict z, int incz, const float *restrict x,
            int incx, const float *restrict y, int incy);

void vrecp_64 (long n, double *restrict y, long incy, const double *restrict x,
               long incx, const double *restrict y, int incy);
void vrecpf_64(long n, float *restrict y, long incy, const float *restrict x,
               long incx, const float *restrict y, int incy);
```

Given input vector **x** and a result vector **y**, this function computes

$$\mathbf{y} = 1/\mathbf{x}$$

### 15.9.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF REAL** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 16 Complex Numbers

## 16.1 `vcreal` - Vector complex real compoment Re(**x**)

```
# include <mvec.h>

void vcreal (int n, double *restrict y, int incy, const double complex *restrict x, int incx);
void vcrealf(int n, float *restrict y, int incy, const float complex *restrict x, int incx);

void vcreal_64 (long n, double *restrict y, long incy, const double complex *restrict x, long incx);
void vcrealf_64(long n, float *restrict y, long incy, const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a real result vector **y**, this function computes:

$$\mathbf{y} = \mathrm{Re}(\mathbf{x})$$

Where $\mathbf{x} \in \mathbb{C}$ and $\mathbf{y} \in \mathbb{R}$.

### 16.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 16.2 vcimag - Vector complex imaginary compoment Im(**x**)

```
# include <mvec.h>

void vcimag (int n, double *restrict y, int incy, const double complex *restrict x, int incx);
void vcimagf(int n, float *restrict y, int incy, const float complex *restrict x, int incx);

void vcimag_64 (long n, double *restrict y, long incy, const double complex *restrict x, long incx);
void vcimagf_64(long n, float *restrict y, long incy, const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a real result vector **y**, this function computes:

$$\mathbf{y} = \mathrm{Im}(\mathbf{x})$$

Where $\mathbf{x} \in \mathbb{C}$ and $\mathbf{y} \in \mathbb{R}$.

### 16.2.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 16.3 vcabs - Vector complex absolute value |x|

```
# include <mvec.h>

void vcabs (int n, double *restrict y, int incy, const double complex *restrict x, int incx);
void vcabsf(int n, float *restrict y, int incy, const float complex *restrict x, int incx);

void vcabs_64 (long n, double *restrict y, long incy, const double complex *restrict x, long incx);
void vcabsf_64(long n, float *restrict y, long incy, const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = |\mathbf{x}|$$

Where $\mathbf{x} \in \mathbb{C}$ and $\mathbf{y} \in \mathbb{R}$.

### 16.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 16.4 vcarg - Vector complex argument arg(**x**)

```
# include <mvec.h>

void vcarg (int n, double *restrict y, int incy, const double complex *restrict x, int incx);
void vcargf(int n, float *restrict y, int incy, const float complex *restrict x, int incx);

void vcarg_64 (long n, double *restrict y, long incy, const double complex *restrict x, long incx);
void vcargf_64(long n, float *restrict y, long incy, const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \arg(\mathbf{x})$$

Where $\mathbf{x} \in \mathbb{C}$ and $\mathbf{y} \in \mathbb{R}$.

### 16.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF REAL** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 16.5 vconj - Vector complex conjugate $\overline{\mathbf{x}}$

```
# include <mvec.h>

void vconj (int n, double complex *restrict y, int incy,
            const double complex *restrict x, int incx);
void vconjf(int n, float complex *restrict y, int incy,
            const float complex *restrict x, int incx);

void vconj_64 (long n, double complex *restrict y, long incy,
               const double complex *restrict x, long incx);
void vconjf_64(long n, float complex *restrict y, long incy,
               const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \overline{\mathbf{x}}$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 16.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 16.6 vcproj - Vector complex Riemann sphere projection proj(**x**)

```
# include <mvec.h>

void vcproj (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vcprojf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);


void vcproj_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vcprojf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \text{proj}(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 16.6.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 17 Complex Exponentials and Logarithms

## 17.1 `vcexp` - Vector complex exponentiation exp(**x**)

```
# include <mvec.h>

void vcexp (int n, double complex *restrict y, int incy,
            const double complex *restrict x, int incx);
void vcexpf(int n, float complex *restrict y, int incy,
            const float complex *restrict x, int incx);

void vcexp_64 (long n, double complex *restrict y, long incy,
               const double complex *restrict x, long incx);
void vcexpf_64(long n, float complex *restrict y, long incy,
               const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \exp(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 17.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

`Y` - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 17.2 vclog - Vector complex logarithm log(**x**)

```
# include <mvec.h>

void vclog (int n, double complex *restrict y, int incy,
            const double complex *restrict x, int incx);
void vclogf(int n, float complex *restrict y, int incy,
            const float complex *restrict x, int incx);

void vclog_64 (long n, double complex *restrict y, long incy,
               const double complex *restrict x, long incx);
void vclogf_64(long n, float complex *restrict y, long incy,
               const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \log(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 17.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 17.3 vcpow - Vector complex power $\mathbf{x}^{\mathbf{y}}$

```
# include <mvec.h>

void vcpow (int n, double complex *restrict z, int incz, const double complex *restrict x,
            int incx, const double complex *restrict y, int incy);
void vcpowf(int n, float complex *restrict z, int incz, const float complex *restrict x,
            int incx, const float complex *restrict y, int incy);

void vcpow_64 (long n, double complex *restrict z, long incz, const double complex *restrict x,
               long incx, const double complex *restrict y, long incy);
void vcpowf_64(long n, float complex *restrict z, long incz, const float complex *restrict x,
               long incx, const float complex *restrict y, long incy);
```

Given an input vectors $\mathbf{x}$ and $\mathbf{y}$ as well as a result vector $\mathbf{z}$, this function computes:

$$\mathbf{z} = \mathbf{x}^{\mathbf{y}}$$

Where $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{C}$.

### 17.3.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$.
*CONSTRAINT:* $n \geq 1$.

`Z` - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{z}$.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$ or $\mathbf{y}$.

`INCZ` - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{z}$.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{z}$ or $\mathbf{y}$.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`Y` - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{z}$ or $\mathbf{x}$.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 17.4 vcpows - Vector complex power scalar exponent $\mathbf{x}^y$

```
# include <mvec.h>

void vcpows (int n, double complex *restrict z, int incz, const double complex *restrict x,
             int incx, double complex y);
void vcpowsf(int n, float complex *restrict z, int incz, const float complex *restrict x,
             int incx, float complex y);

void vcpows_64 (long n, double complex *restrict z, long incz, const double complex *restrict x,
                long incx, double complex y);
void vcpowsf_64(long n, float complex *restrict z, long incz, const float complex *restrict x,
                long incx, float complex y);
```

Given an input vectors **x** and a scalar value $y$ as well as a result vector **z**, this function computes:

$$\mathbf{z} = \mathbf{x}^y$$

### 17.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

z - **ARRAY OF COMPLEX** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **COMPLEX** *ENTRY:* Exponentiation value $y$.

## 17.5 vccis - Vector complex Euler's Formula exp(*i***x**)

```
# include <mvec.h>

void vccis (int n, double complex *restrict y, int incy,
            const double complex *restrict x, int incx);
void vccisf(int n, float complex *restrict y, int incy,
            const float complex *restrict x, int incx);

void vccis_64 (long n, double complex *restrict y, long incy,
               const double complex *restrict x, long incx);
void vccisf_64(long n, float complex *restrict y, long incy,
               const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \exp(i\mathbf{x}) = \cos(\mathbf{x}) + i\sin(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 17.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 18 Complex Roots

## 18.1 vcsqrt - Vector complex square root $\sqrt{\mathbf{x}}$

```
# include <mvec.h>

void vcsqrt (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vcsqrtf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);

void vcsqrt_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vcsqrtf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \sqrt{\mathbf{x}}$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 18.1.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 19 Complex Trigonometric Functions

## 19.1 vcsin - Vector complex sine sin(**x**)

```
# include <mvec.h>

void vcsin (int n, double complex *restrict y, int incy,
            const double complex *restrict x, int incx);
void vcsinf(int n, float complex *restrict y, int incy,
            const float complex *restrict x, int incx);

void vcsin_64 (long n, double complex *restrict y, long incy,
               const double complex *restrict x, long incx);
void vcsinf_64(long n, float complex *restrict y, long incy,
               const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sin(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 19.1.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 19.2 vccos - Vector complex cosine cos(**x**)

```
# include <mvec.h>

void vccos (int n, double complex *restrict y, int incy,
            const double complex *restrict x, int incx);
void vccosf(int n, float complex *restrict y, int incy,
            const float complex *restrict x, int incx);

void vccos_64 (long n, double complex *restrict y, long incy,
               const double complex *restrict x, long incx);
void vccosf_64(long n, float complex *restrict y, long incy,
               const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cos(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 19.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 19.3 vctan - Vector complex tangent tan(**x**)

```
# include <mvec.h>

void vctan (int n, double complex *restrict y, int incy,
            const double complex *restrict x, int incx);
void vctanf(int n, float complex *restrict y, int incy,
            const float complex *restrict x, int incx);

void vctan_64 (long n, double complex *restrict y, long incy,
               const double complex *restrict x, long incx);
void vctanf_64(long n, float complex *restrict y, long incy,
               const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tan(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 19.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 19.4 vcasin - Vector complex arcsine $\sin^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vcasin (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vcasinf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);

void vcasin_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vcasinf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \sin^{-1}(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 19.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 19.5 vcacos - Vector complex arccosine $\cos^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vcacos (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vcacosf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);

void vcacos_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vcacosf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \cos^{-1}(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 19.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 19.6 vcatan - Vector complex arctangent $\tan^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vcatan (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vcatanf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);

void vcatan_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vcatanf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \tan^{-1}(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 19.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 20 Complex Hypergeometric Functions

## 20.1 vcsinh - Vector complex hyperbolic sine sinh(**x**)

```
# include <mvec.h>

void vcsinh (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vcsinhf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);

void vcsinh_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vcsinhf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \sinh(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 20.1.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 20.2 vccosh - Vector complex hyperbolic cosine cosh(**x**)

```
# include <mvec.h>

void vccosh (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vccoshf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);

void vccosh_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vccoshf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \cosh(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 20.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 20.3 vctanh - Vector complex hyperbolic tangent tanh(**x**)

```
# include <mvec.h>

void vctanh (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx);
void vctanhf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx);

void vctanh_64 (long n, double complex *restrict y, long incy,
                const double complex *restrict x, long incx);
void vctanhf_64(long n, float complex *restrict y, long incy,
                const float complex *restrict x, long incx);
```

Given a complex input vector **x** and a result vector **y**, this function computes:

$$\mathbf{y} = \tanh(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 20.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 20.4 vcasinh - Vector complex hyperbolic arcsine $\sinh^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vcasinh (int n, double complex *restrict y, int incy,
              const double complex *restrict x, int incx);
void vcasinhf(int n, float complex *restrict y, int incy,
              const float complex *restrict x, int incx);

void vcasinh_64 (long n, double complex *restrict y, long incy,
                 const double complex *restrict x, long incx);
void vcasinhf_64(long n, float complex *restrict y, long incy,
                 const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \sinh^{-1}(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 20.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 20.5 vcacosh - Vector complex hyperbolic arccosine $\cosh^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vcacosh (int n, double complex *restrict y, int incy,
              const double complex *restrict x, int incx);
void vcacoshf(int n, float complex *restrict y, int incy,
              const float complex *restrict x, int incx);

void vcacosh_64 (long n, double complex *restrict y, long incy,
                 const double complex *restrict x, long incx);
void vcacoshf_64(long n, float complex *restrict y, long incy,
                 const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \cosh^{-1}(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 20.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 20.6 vcatanh - Vector complex hyperbolic arctangent $\tanh^{-1}(\mathbf{x})$

```
# include <mvec.h>

void vcatanh (int n, double complex *restrict y, int incy,
              const double complex *restrict x, int incx);
void vcatanhf(int n, float complex *restrict y, int incy,
              const float complex *restrict x, int incx);

void vcatanh_64 (long n, double complex *restrict y, long incy,
                 const double complex *restrict x, long incx);
void vcatanhf_64(long n, float complex *restrict y, long incy,
                 const float complex *restrict x, long incx);
```

Given a complex input vector $\mathbf{x}$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \tanh^{-1}(\mathbf{x})$$

Where $\mathbf{x}, \mathbf{y} \in \mathbb{C}$.

### 20.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

# 21 Complex Arithmetic

## 21.1 vcadd - Vector complex addition x + y

```
# include <mvec.h>

void vcadd (int n, double complex *restrict z, int incz, const double complex *restrict x,
            int incx, const double complex *restrict y, int incy);
void vcaddf(int n, float complex *restrict z, int incz, const float complex *restrict x,
            int incx, const float complex *restrict y, int incy);

void vcadd_64 (long n, double complex *restrict z, long incz, const double complex *restrict x,
               long incx, const double complex *restrict y, long incy);
void vcaddf_64(long n, float complex *restrict z, long incz, const float complex *restrict x,
               long incx, const float complex *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{x} + \mathbf{y}$$

### 21.1.1 Parameters

`N` - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

`Z` - **ARRAY OF COMPLEX** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

`INCZ` - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`X` - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

`INCX` - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

`Y` - **ARRAY OF COMPLEX** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

`INCY` - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 21.2 vcsadd - Vector complex scalar subtraction $\mathbf{x}+\alpha$

```
# include <mvec.h>

void vcsadd (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx, double complex a);
void vcsaddf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx, float complex a);

void vcsadd_64 (long n, double complex *restrict y, long incy,
                double complex *restrict x, long incx, double complex a);
void vcsaddf_64(long n, float complex *restrict y, long incy,
                float complex *restrict x, long incx, float complex a);
```

Given an input vector $\mathbf{x}$ and a scalar constant $\alpha$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \mathbf{x} + \alpha$$

### 21.2.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **COMPLEX** *ENTRY:* Scalar constant $\alpha$.

## 21.3 vcsub - Vector complex subtraction $\mathbf{x}-\mathbf{y}$

```
# include <mvec.h>

void vcsub (int n, double complex *restrict z, int incz, const double complex *restrict x,
            int incx, const double complex *restrict y, int incy);
void vcsubf(int n, float complex *restrict z, int incz, const float complex *restrict x,
            int incx, const float complex *restrict y, int incy);

void vcsub_64 (long n, double complex *restrict z, long incz, const double complex *restrict x,
               long incx, const double complex *restrict y, long incy);
void vcsubf_64(long n, float complex *restrict z, long incz, const float complex *restrict x,
               long incx,  const float complex *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{x} - \mathbf{y}$$

### 21.3.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF COMPLEX** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF COMPLEX** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 21.4 vcssub - Vector complex scalar subtraction $\mathbf{x} - \alpha$

```
# include <mvec.h>

void vcssub (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx, double complex a);
void vcssubf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx, float complex a);

void vcssub_64 (long n, double complex *restrict y, long incy,
                double complex *restrict x, long incx, double complex a);
void vcssubf_64(long n, float complex *restrict y, long incy,
                float complex *restrict x, long incx, float complex a);
```

Given an input vector $\mathbf{x}$ and a scalar constant $\alpha$ and a result vector $\mathbf{y}$, this function computes:

$$\mathbf{y} = \mathbf{x} - \alpha$$

### 21.4.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of $\mathbf{x}$ and $\mathbf{y}$.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector $\mathbf{y}$.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{x}$.

INCY - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{y}$.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector $\mathbf{x}$.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array $\mathbf{y}$.

INCX - **INTEGER** *ENTRY:* Stride for the vector $\mathbf{x}$.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **COMPLEX** *ENTRY:* Scalar constant $\alpha$.

## 21.5 vcmul - Vector complex multiplication xy

```
# include <mvec.h>

void vcmul (int n, double complex *restrict z, int incz, const double complex *restrict x,
            int incx, const double complex *y, int incy);
void vcmulf(int n, float complex *restrict z, int incz, const float complex *restrict x,
            int incx, const float complex *y, int incy);

void vcmul_64 (long n, double complex *restrict z, long incz, const double complex *restrict x,
               long incx, const double complex *restrict y, long incy);
void vcmulf_64(long n, float complex *restrict z, long incz, const float complex *restrict x,
               long incx, const float complex *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{xy}$$

### 21.5.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF COMPLEX** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF COMPLEX** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 21.6 vcsmul - Vector complex scalar multiplication $\alpha\mathbf{x}$

```
# include <mvec.h>

void vcsmul (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx, double complex a);
void vcsmulf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx, float complex a);

void vcsmul_64 (long n, double complex *restrict y, long incy,
                double complex *restrict x, long incx, double complex a);
void vcsmulf_64(long n, float complex *restrict y, long incy,
                float complex *restrict x, long incx, float complex a);
```

Given an input vector **x** and a scalar constant $\alpha$ and a result vector **y**, this function computes:

$$\mathbf{y} = \alpha\mathbf{x}$$

### 21.6.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **COMPLEX** *ENTRY:* Scalar constant $\alpha$.

## 21.7 vcmulc - Vector complex conjugate multiplication $\mathbf{x}\bar{\mathbf{y}}$

```
# include <mvec.h>

void vcmulc (int n, double complex *restrict z, int incz, const double complex *restrict x,
             int incx, const double complex *restrict y, int incy);
void vcmulcf(int n, float complex *restrict z, int incz, const float complex *restrict x,
             int incx, const float complex *restrict y, int incy);

void vcmulc_64 (long n, double complex *restrict z, long incz, const double complex *restrict x,
                long incx, const double complex *restrict y, long incy);
void vcmulcf_64(long n, float complex *restrict z, long incz, const float complex *restrict x,
                long incx, const float complex *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{x}\bar{\mathbf{y}}$$

### 21.7.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF COMPLEX** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF COMPLEX** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 21.8 vcdiv - Vector complex division x/y

```
# include <mvec.h>

void vcdiv (int n, double complex *restrict z, int incz, const double complex *restrict x,
            int incx, const double complex *restrict y, int incy);
void vcdivf(int n, float complex *restrict z, int incz, const float complex *restrict x,
            int incx, const float complex *restrict y, int incy);

void vcdiv_64 (long n, double complex *restrict z, long incz, const double complex *restrict x,
               long incx, const double complex *restrict y, long incy);
void vcdivf_64(long n, float complex *restrict z, long incz, const float complex *restrict x,
               long incx, const float complex *restrict y, long incy);
```

Given input vectors **x** and **y** and a result vector **z**, this function computes

$$\mathbf{z} = \mathbf{x}/\mathbf{y}$$

### 21.8.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x**, **y** and **z**.
*CONSTRAINT:* $n \geq 1$.

Z - **ARRAY OF COMPLEX** *EXIT:* Result vector **z**.
*CONSTRAINT:* Must contain $n \times$ incz elements.
*CONSTRAINT:* Must not overlap with array **x** or **y**.

INCZ - **INTEGER** *ENTRY:* Stride for the vector **z**.
*CONSTRAINT:* incz $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **z** or **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

Y - **ARRAY OF COMPLEX** *ENTRY:* Input vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **z** or **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

## 21.9 vcsdiv - Vector complex scalar division **x**/$\alpha$

```
# include <mvec.h>

void vcsdiv (int n, double complex *restrict y, int incy,
             const double complex *restrict x, int incx, double complex a);
void vcsdivf(int n, float complex *restrict y, int incy,
             const float complex *restrict x, int incx, float complex a);

void vcsdiv_64 (long n, double complex *restrict y, long incy,
                double complex *restrict x, long incx, double complex a);
void vcsdivf_64(long n, float complex *restrict y, long incy,
                float complex *restrict x, long incx, float complex a);
```

Given an input vector **x** and a scalar constant $\alpha$ and a result vector **y**, this function computes:

$$\mathbf{y} = \mathbf{x}/\alpha$$

### 21.9.1 Parameters

N - **INTEGER** *ENTRY:* Number of elements of **x** and **y**.
*CONSTRAINT:* $n \geq 1$.

Y - **ARRAY OF COMPLEX** *EXIT:* Result vector **y**.
*CONSTRAINT:* Must contain $n \times$ incy elements.
*CONSTRAINT:* Must not overlap with array **x**.

INCY - **INTEGER** *ENTRY:* Stride for the vector **y**.
*CONSTRAINT:* incy $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

X - **ARRAY OF COMPLEX** *ENTRY:* Input vector **x**.
*CONSTRAINT:* Must contain $n \times$ incx elements.
*CONSTRAINT:* Must not overlap with array **y**.

INCX - **INTEGER** *ENTRY:* Stride for the vector **x**.
*CONSTRAINT:* incx $\neq 0$.
*BEHAVIOR:* A negative stride will traverse the array in reverse.

A - **COMPLEX** *ENTRY:* Scalar constant $\alpha$.

# 22 Acknowledgements

Parts of this product include or are derived from code under the following licences:

```
/*
 * Copyright (c) 2003, 2017, 2023 Steven G. Kargl
 * Copyright (c) 2005, 2011 David Schultz <das@FreeBSD.ORG>
 * Copyright (c) 2005 Bruce D. Evans and Steven G. Kargl
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

/*
 * Copyright (c) 2008 Stephen L. Moshier <steve@moshier.net>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

------------------------------------------------------------------------
Copyright © 2005-2020 Rich Felker, et al.
Copyright © 2013 Szabolcs Nagy

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:
```