Math Vector Library 1.1
Reference Manual (Fortran)
DD-00002-111

Jan Adelsbach

February 17, 2025

Contents

Copyright © Adelsbach

Copyright © Adelsbach

Copyright © Adelsbach

Copyright © Adelsbach

Copyright © Adelsbach

1 About this Guide

1.1 Legal Information

Copyright ©2024-2025 Adelsbach UG (haftungsbeschränkt). All Rights Reserved.
Copyright ©2023-2025 Jan Adelsbach. All Rights Reserved.
From herein referred to as *Adelsbach*.

This document may not be reproduced without written persmission by Adelsbach.

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 Fortran programming language families.

1.4 Audience for This Guide

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

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.

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.

Copyright © Adelsbach

3 Utility

3.1 `mvecver` - Version query

```
subroutine mvecver(major, 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

Copyright © Adelsbach

4 Rounding

4.1 `vfloor` - Vector round down

```
subroutine vfloor (n, y, incy, x, incx);
subroutine vfloorf(n, y, incy, x, incx);

subroutine vfloor_64 (n, y, incy, x, incx);
subroutine vfloorf_64(n, y, incy, x, 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.

Copyright © Adelsbach

4.2 vceil - Vector round up

```
subroutine vceil (n, y, incy, x, incx);
subroutine vceilf(n, y, incy, x, incx);

subroutine vceil_64 (n, y, incy, x, incx);
subroutine vceilf_64(n, y, incy, x, 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.

Copyright © Adelsbach

4.3 vtrunc - Vector truncate

```
subroutine vtrunc (n, y, incy, x, incx);
subroutine vtruncf(n, y, incy, x, incx);

subroutine vtrunc_64 (n, y, incy, x, incx);
subroutine vtruncf_64(n, y, incy, x, 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.

Copyright © Adelsbach

4.4 vround - Vector rounding

```
subroutine vround (n, y, incy, x, incx);
subroutine vroundf(n, y, incy, x, incx);

subroutine vround_64 (n, y, incy, x, incx);
subroutine vroundf_64(n, y, incy, x, 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.

Copyright © Adelsbach

5 Roots

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

```
subroutine vsqrt (n, y, incy, x, incx);
subroutine vsqrtf(n, y, incy, x, incx);

subroutine vsqrt_64 (n, y, incy, x, incx);
subroutine vsqrtf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vrsqrt (n, y, incy, x, incx);
subroutine vrsqrtf(n, y, incy, x, incx);

subroutine vrsqrt_64 (n, y, incy, x, incx);
subroutine vrsqrtf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vcbrt (n, y, incy, x, incx);
subroutine vcbrtf(n, y, incy, x, incx);

subroutine vcbrt_64 (n, y, incy, x, incx);
subroutine vcbrtf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vrcbrt (n, y, incy, x, incx);
subroutine vrcbrtf(n, y, incy, x, incx);

subroutine vrcbrt_64 (n, y, incy, x, incx);
subroutine vrcbrtf_64(n, y, incy, x, 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.

Copyright © Adelsbach

6 Trigonometric Functions

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

```
subroutine vsin (n, y, incy, x, incx);
subroutine vsinf(n, y, incy, x, incx);

subroutine vsin_64 (n, y, incy, x, incx);
subroutine vsinf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vcos (n, y, incy, x, incx);
subroutine vcosf(n, y, incy, x, incx);

subroutine vcos_64 (n, y, incy, x, incx);
subroutine vcosf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vtan (n, y, incy, x, incx);
subroutine vtanf(n, y, incy, x, incx);

subroutine vtan_64 (n, y, incy, x, incx);
subroutine vtanf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vasin (n, y, incy, x, incx);
subroutine vasinf(n, y, incy, x, incx);

subroutine vasin_64 (n, y, incy, x, incx);
subroutine vasinf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vacos (n, y, incy, x, incx);
subroutine vacosf(n, y, incy, x, incx);

subroutine vacos_64 (n, y, incy, x, incx);
subroutine vacosf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vatan (n, y, incy, x, incx);
subroutine vatanf(n, y, incy, x, incx);

subroutine vatan_64 (n, y, incy, x, incx);
subroutine vatanf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vatan2 (n, z, incz, x, incx, y, incy);
subroutine vatan2f(n, z, incz, x, incx, y, incy);

subroutine vatan2_64 (n, z, incz, x, incx, y, incy);
subroutine vatan2f_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

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

```
subroutine vsind (n, y, incy, x, incx);
subroutine vsindf(n, y, incy, x, incx);

subroutine vsind_64 (n, y, incy, x, incx);
subroutine vsindf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vcosd (n, y, incy, x, incx);
subroutine vcosdf(n, y, incy, x, incx);

subroutine vcosd_64 (n, y, incy, x, incx);
subroutine vcosdf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vtand (n, y, incy, x, incx);
subroutine vtandf(n, y, incy, x, incx);

subroutine vtand_64 (n, y, incy, x, incx);
subroutine vtandf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vasind (n, y, incy, x, incx);
subroutine vasindf(n, y, incy, x, incx);

subroutine vasind_64 (n, y, incy, x, incx);
subroutine vasindf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vacosd (n, y, incy, x, incx);
subroutine vacosdf(n, y, incy, x, incx);

subroutine vacosd_64 (n, y, incy, x, incx);
subroutine vacosdf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vatand (n, y, incy, x, incx);
subroutine vatandf(n, y, incy, x, incx);

subroutine vatand_64 (n, y, incy, x, incx);
subroutine vatandf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vsinpi (n, y, incy, x, incx);
subroutine vsinpif(n, y, incy, x, incx);

subroutine vsinpi_64 (n, y, incy, x, incx);
subroutine vsinpif_64(n, y, incy, x, 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.

Copyright © Adelsbach

6.15 vcospi - Vector half-cycle cosine cos(πx)

```
subroutine vcospi (n, y, incy, x, incx);
subroutine vcospif(n, y, incy, x, incx);

subroutine vcospi_64 (n, y, incy, x, incx);
subroutine vcospif_64(n, y, incy, x, 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.

Copyright © Adelsbach

6.16 vtanpi - Vector half-cycle tangent tan(π**x**)

```
subroutine vtanpi (n, y, incy, x, incx);
subroutine vtanpif(n, y, incy, x, incx);

subroutine vtanpi_64 (n, y, incy, x, incx);
subroutine vtanpif_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vasinpi (n, y, incy, x, incx);
subroutine vasinpif(n, y, incy, x, incx);

subroutine vasinpi_64 (n, y, incy, x, incx);
subroutine vasinpif_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vacospi (n, y, incy, x, incx);
subroutine vacospif(n, y, incy, x, incx);

subroutine vacospi_64 (n, y, incy, x, incx);
subroutine vacospif_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vatanpi (n, y, incy, x, incx);
subroutine vatanpif(n, y, incy, x, incx);

subroutine vatanpi_64 (n, y, incy, x, incx);
subroutine vatanpif_64(n, y, incy, x, 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.

Copyright © Adelsbach

7 Hypergeometric Functions

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

```
subroutine vsinh (n, y, incy, x, incx);
subroutine vsinhf(n, y, incy, x, incx);

subroutine vsinh_64 (n, y, incy, x, incx);
subroutine vsinhf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vcosh (n, y, incy, x, incx);
subroutine vcoshf(n, y, incy, x, incx);

subroutine vcosh_64 (n, y, incy, x, incx);
subroutine vcoshf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vtanh (n, y, incy, x, incx);
subroutine vtanhf(n, y, incy, x, incx);

subroutine vtanh_64 (n, y, incy, x, incx);
subroutine vtanhf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vasinh (n, y, incy, x, incx);
subroutine vasinhf(n, y, incy, x, incx);

subroutine vasinh_64 (n, y, incy, x, incx);
subroutine vasinhf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vacosh (n, y, incy, x, incx);
subroutine vacoshf(n, y, incy, x, incx);

subroutine vacosh_64 (n, y, incy, x, incx);
subroutine vacoshf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vatanh (n, y, incy, x, incx);
subroutine vatanhf(n, y, incy, x, incx);

subroutine vatanh_64 (n, y, incy, x, incx);
subroutine vatanhf_64(n, y, incy, x, 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.

Copyright © Adelsbach

8 Exponentials and Logarithms

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

```
subroutine vexp (n, y, incy, x, incx);
subroutine vexpf(n, y, incy, x, incx);

subroutine vexp_64 (n, y, incy, x, incx);
subroutine vexpf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vexpm1 (n, y, incy, x, incx);
subroutine vexpm1f(n, y, incy, x, incx);

subroutine vexpm1_64 (n, y, incy, x, incx);
subroutine vexpm1f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vexp2 (n, y, incy, x, incx);
subroutine vexp2f(n, y, incy, x, incx);

subroutine vexp2_64 (n, y, incy, x, incx);
subroutine vexp2f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vexp2m1 (n, y, incy, x, incx);
subroutine vexp2m1f(n, y, incy, x, incx);

subroutine vexp2m1_64 (n, y, incy, x, incx);
subroutine vexp2m1f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vexp10 (n, y, incy, x, incx);
subroutine vexp10f(n, y, incy, x, incx);

subroutine vexp10_64 (n, y, incy, x, incx);
subroutine vexp10f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vexp10m1 (n, y, incy, x, incx);
subroutine vexp10m1f(n, y, incy, x, incx);

subroutine vexp10m1_64 (n, y, incy, x, incx);
subroutine vexp10m1f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vlog (n, y, incy, x, incx);
subroutine vlogf(n, y, incy, x, incx);

subroutine vlog_64 (n, y, incy, x, incx);
subroutine vlogf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vlog2 (n, y, incy, x, incx);
subroutine vlog2f(n, y, incy, x, incx);

subroutine vlog2_64 (n, y, incy, x, incx);
subroutine vlog2f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vlog10 (n, y, incy, x, incx);
subroutine vlog10f(n, y, incy, x, incx);

subroutine vlog10_64 (n, y, incy, x, incx);
subroutine vlog10f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vlog1p (n, y, incy, x, incx);
subroutine vlog1pf(n, y, incy, x, incx);

subroutine vlog1p_64 (n, y, incy, x, incx);
subroutine vlog1pf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vpow (n, z, incz, x, incx, y, incy);
subroutine vpowf(n, z, incz, x, incx, y, incy);

subroutine vpow_64 (n, z, incz, x, incx, y, incy);
subroutine vpowf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

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

```
subroutine vpows (n, z, incz, x, incx, y);
subroutine vpowsf(n, z, incz, x, incx, y);

subroutine vpows_64 (n, z, incz, x, incx, y);
subroutine vpowsf_64(n, z, incz, x, incx, 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.

Copyright © Adelsbach

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

```
subroutine vmod (n, z, incz, x, incx, y, incy);
subroutine vmodf(n, z, incz, x, incx, y, incy);

subroutine vmod_64 (n, z, incz, x, incx, y, incy);
subroutine vmodf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

9 Error Functions

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

```
subroutine verf (n, y, incy, x, incx);
subroutine verff(n, y, incy, x, incx);

subroutine verf_64 (n, y, incy, x, incx);
subroutine verff_64(n, y, incy, x, 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.

Copyright © Adelsbach

9.2 verfinv - Vector inverse error function $\text{erf}^{-1}(\mathbf{x})$

```
subroutine verfinv (n, y, incy, x, incx);
subroutine verfinvf(n, y, incy, x, incx);

subroutine verfinv_64 (n, y, incy, x, incx);
subroutine verfinvf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine verfc (n, y, incy, x, incx);
subroutine verfcf(n, y, incy, x, incx);

subroutine verfc_64 (n, y, incy, x, incx);
subroutine verfcf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine verfcinv (n, y, incy, x, incx);
subroutine verfcinvf(n, y, incy, x, incx);

subroutine verfcinv_64 (n, y, incy, x, incx);
subroutine verfcinvf_64(n, y, incy, x, 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.

Copyright © Adelsbach

10 Bessel Functions

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

```
subroutine vbesj0 (n, y, incy, x, incx);
subroutine vbesj0f(n, y, incy, x, incx);

subroutine vbesj0_64 (n, y, incy, x, incx);
subroutine vbesj0f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vbesy0 (n, y, incy, x, incx);
subroutine vbesy0f(n, y, incy, x, incx);

subroutine vbesy0_64 (n, y, incy, x, incx);
subroutine vbesy0f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vbesj1 (n, y, incy, x, incx);
subroutine vbesj1f(n, y, incy, x, incx);

subroutine vbesj1_64 (n, y, incy, x, incx);
subroutine vbesj1f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vbesy1 (n, y, incy, x, incx);
subroutine vbesy1f(n, y, incy, x, incx);

subroutine vbesy1_64 (n, y, incy, x, incx);
subroutine vbesy1f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vbesjn (n, y, incy, k, x, incx);
subroutine vbesjnf(n, y, incy, k, x, incx);

subroutine vbesjn_64 (n, y, incy, k, x, incx);
subroutine vbesjnf_64(n, y, incy, k, x, 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.

Copyright © Adelsbach

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

```
subroutine vbesyn (n, y, incy, k, x, incx);
subroutine vbesynf(n, y, incy, k, x, incx);

subroutine vbesyn_64 (n, y, incy, k, x, incx);
subroutine vbesynf_64(n, y, incy, k, x, 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.

Copyright © Adelsbach

11 Gamma Function Related

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

```
subroutine vlgamma (n, y, incy, x, incx);
subroutine vlgammaf(n, y, incy, x, incx);

subroutine vlgamma_64 (n, y, incy, x, incx);
subroutine vlgammaf_64(n, y, incy, x, 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.

Copyright © Adelsbach

12 Einstein Functions

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

```
subroutine veinst1 (n, y, incy, x, incx);
subroutine veinst1f(n, y, incy, x, incx);

subroutine veinst1_64 (n, y, incy, x, incx);
subroutine veinst1f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine veinst2 (n, y, incy, x, incx);
subroutine veinst2f(n, y, incy, x, incx);

subroutine veinst2_64 (n, y, incy, x, incx);
subroutine veinst2f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine veinst3 (n, y, incy, x, incx);
subroutine veinst3f(n, y, incy, x, incx);

subroutine veinst3_64 (n, y, incy, x, incx);
subroutine veinst3f_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine veinst4 (n, y, incy, x, incx);
subroutine veinst4f(n, y, incy, x, incx);

subroutine veinst4_64 (n, y, incy, x, incx);
subroutine veinst4f_64(n, y, incy, x, 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.

Copyright © Adelsbach

13 Integrals

13.1 `vsi` - Vector Sine Integral Si(**x**)

```
subroutine vsi (n, y, incy, x, incx);
subroutine vsif(n, y, incy, x, incx);

subroutine vsi_64 (n, y, incy, x, incx);
subroutine vsif_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vci (n, y, incy, x, incx);
subroutine vcif(n, y, incy, x, incx);

subroutine vci_64 (n, y, incy, x, incx);
subroutine vcif_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vti2 (n, y, incy, x, incx);
subroutine vti2f(n, y, incy, x, incx);

subroutine vti2_64 (n, y, incy, x, incx);
subroutine vti2f_64(n, y, incy, x, 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.

Copyright © Adelsbach

14 Other Functions

14.1 `vabs` - Vector absolute value |**x**|

```
subroutine vabs (n, y, incy, x, incx);
subroutine vabsf(n, y, incy, x, incx);

subroutine vabs_64 (n, y, incy, x, incx);
subroutine vabsf_64(n, y, incy, x, 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.

Copyright © Adelsbach

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

```
subroutine vhypot (n, z, incz, x, incx, y, incy);
subroutine vhypotf(n, z, incz, x, incx, y, incy);

subroutine vhypot_64 (n, z, incz, x, incx, y, incy);
subroutine vhypotf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

14.3 vrem - Vector remainder

```
subroutine vrem (n, z, incz, x, incx, y, incy);
subroutine vremf(n, z, incz, x, incx, y, incy);

subroutine vrem_64 (n, z, incz, x, incx, y, incy);
subroutine vremf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

15 Arithmetic Functions

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

```
subroutine vadd (n, z, incz, x, incx, y, incy);
subroutine vaddf(n, z, incz, x, incx, y, incy);

subroutine vadd_64 (n, z, incz, x, incx, y, incy);
subroutine vaddf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

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

```
subroutine vsadd (n, y, incy, x, incx, a);
subroutine vsaddf(n, y, incy, x, incx, a);

subroutine vsadd_64 (n, y, incy, x, incx, a);
subroutine vsaddf_64(n, y, incy, x, incx, a);
```

Given an input vector $\mathbf{x}$ and a scalar constant α 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 α.

Copyright © Adelsbach

15.3 vsub - Vector subtraction x − y

```
subroutine vsub (n, z, incz, x, incx, y, incy);
subroutine vsubf(n, z, incz, x, incx, y, incy);

subroutine vsub_64 (n, z, incz, x, incx, y, incy);
subroutine vsubf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

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

```
subroutine vssub (n, y, incy, x, incx, a);
subroutine vssubf(n, y, incy, x, incx, a);

subroutine vssub_64 (n, y, incy, x, incx, a);
subroutine vssubf_64(n, y, incy, x, incx, a);
```

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

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

15.4.1 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.

A - **REAL** *ENTRY:* Scalar constant α.

Copyright © Adelsbach

15.5 vmul - Vector multiplication xy

```
subroutine vmul (n, z, incz, x, incx, y, incy);
subroutine vmulf(n, z, incz, x, incx, y, incy);

subroutine vmul_64 (n, z, incz, x, incx, y, incy);
subroutine vmulf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

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

```
subroutine vsmul (n, y, incy, x, incx, a);
subroutine vsmulf(n, y, incy, x, incx, a);

subroutine vsmul_64 (n, y, incy, x, incx, a);
subroutine vsmulf_64(n, y, incy, x, incx, a);
```

Given an input vector **x** and a scalar constant α 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 α.

Copyright © Adelsbach

15.7 vdiv - Vector division x/y

```
subroutine vdiv (n, z, incz, x, incx, y, incy);
subroutine vdivf(n, z, incz, x, incx, y, incy);

subroutine vdiv_64 (n, z, incz, x, incx, y, incy);
subroutine vdivf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

15.8 vsdiv - Vector scalar division **x**/α

```
subroutine vsdiv (n, y, incy, x, incx, a);
subroutine vsdivf(n, y, incy, x, incx, a);

subroutine vsdiv_64 (n, y, incy, x, incx, a);
subroutine vsdivf_64(n, y, incy, x, incx, a);
```

Given an input vector **x** and a scalar constant α 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 α.
CONSTRAINT: $\alpha \neq 0$.

Copyright © Adelsbach

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

```
subroutine vrecp (n, z, incz, x, incx, y, incy);
subroutine vrecpf(n, z, incz, x, incx, y, incy);

subroutine vrecp_64 (n, z, incz, x, incx, y, incy);
subroutine vrecpf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

16 Complex Numbers

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

```
subroutine vcreal (n, y, incy, x, incx);
subroutine vcrealf(n, y, incy, x, incx);

subroutine vcreal_64 (n, y, incy, x, incx);
subroutine vcrealf_64(n, y, incy, x, 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.

Copyright © Adelsbach

16.2 vcimag - Vector complex imaginary compoment Im(**x**)

```
subroutine vcimag (n, y, incy, x, incx);
subroutine vcimagf(n, y, incy, x, incx);

subroutine vcimag_64 (n, y, incy, x, incx);
subroutine vcimagf_64(n, y, incy, x, 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.

Copyright © Adelsbach

16.3 vcabs - Vector complex absolute value |x|

```
subroutine vcabs (n, y, incy, x, incx);
subroutine vcabsf(n, y, incy, x, incx);

subroutine vcabs_64 (n, y, incy, x, incx);
subroutine vcabsf_64(n, y, incy, x, 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.

Copyright © Adelsbach

16.4 vcarg - Vector complex argument arg(**x**)

```
subroutine vcarg (n, y, incy, x, incx);
subroutine vcargf(n, y, incy, x, incx);

subroutine vcarg_64 (n, y, incy, x, incx);
subroutine vcargf_64(n, y, incy, x, 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.

Copyright © Adelsbach

16.5 vconj - Vector complex conjugate $\overline{\mathbf{x}}$

```
subroutine vconj (n, y, incy, x, incx);
subroutine vconjf(n, y, incy, x, incx);

subroutine vconj_64 (n, y, incy, x, incx);
subroutine vconjf_64(n, y, incy, x, incx);
```

Given a complex input vector **x** and a result vector **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 **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.

Copyright © Adelsbach

16.6 vcproj - Vector complex Riemann sphere projection proj(**x**)

```
subroutine vcproj (n, y, incy, x, incx);
subroutine vcprojf(n, y, incy, x, incx);

subroutine vcproj_64 (n, y, incy, x, incx);
subroutine vcprojf_64(n, y, incy, x, 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.

Copyright © Adelsbach

17 Complex Exponentials and Logarithms

17.1 `vcexp` - Vector complex exponentiation exp(**x**)

```
subroutine vcexp (n, y, incy, x, incx);
subroutine vcexpf(n, y, incy, x, incx);

subroutine vcexp_64 (n, y, incy, x, incx);
subroutine vcexpf_64(n, y, incy, x, 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.

Copyright © Adelsbach

17.2 vclog - Vector complex logarithm log(**x**)

```
subroutine vclog (n, y, incy, x, incx);
subroutine vclogf(n, y, incy, x, incx);

subroutine vclog_64 (n, y, incy, x, incx);
subroutine vclogf_64(n, y, incy, x, 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.

Copyright © Adelsbach

17.3 vcpow - Vector complex power $\mathbf{x}^{\mathbf{y}}$

```
subroutine vcpow (n, z, incz, x, incx, y, incy);
subroutine vcpowf(n, z, incz, x, incx, y, incy);

subroutine vcpow_64 (n, z, incz, x, incx, y, incy);
subroutine vcpowf_64(n, z, incz, x, incx, y, incy);
```

Given an input vectors **x** and **y** as well as a result vector **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 **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.

Copyright © Adelsbach

17.4 vcpows - Vector complex power scalar exponent $\mathbf{x}^y$

```
subroutine vcpows (n, z, incz, x, incx, y);
subroutine vcpowsf(n, z, incz, x, incx, y);

subroutine vcpows_64 (n, z, incz, x, incx, y);
subroutine vcpowsf_64(n, z, incz, x, incx, 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.

Copyright © Adelsbach

17.5 vccis - Vector complex Euler's Formula exp($i\mathbf{x}$)

```
subroutine vccis (n, y, incy, x, incx);
subroutine vccisf(n, y, incy, x, incx);

subroutine vccis_64 (n, y, incy, x, incx);
subroutine vccisf_64(n, y, incy, x, 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.

Copyright © Adelsbach

18 Complex Roots

18.1 `vcsqrt` - Vector complex square root $\sqrt{\mathbf{x}}$

```
subroutine vcsqrt (n, y, incy, x, incx);
subroutine vcsqrtf(n, y, incy, x, incx);

subroutine vcsqrt_64 (n, y, incy, x, incx);
subroutine vcsqrtf_64(n, y, incy, x, incx);
```

Given a complex input vector **x** and a result vector **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 **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.

Copyright © Adelsbach

19 Complex Trigonometric Functions

19.1 `vcsin` - Vector complex sine $\sin(\mathbf{x})$

```
subroutine vcsin (n, y, incy, x, incx);
subroutine vcsinf(n, y, incy, x, incx);

subroutine vcsin_64 (n, y, incy, x, incx);
subroutine vcsinf_64(n, y, incy, x, 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.

Copyright © Adelsbach

19.2 vccos - Vector complex cosine cos(**x**)

```
subroutine vccos (n, y, incy, x, incx);
subroutine vccosf(n, y, incy, x, incx);

subroutine vccos_64 (n, y, incy, x, incx);
subroutine vccosf_64(n, y, incy, x, 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.

Copyright © Adelsbach

19.3 vctan - Vector complex tangent tan(**x**)

```
subroutine vctan (n, y, incy, x, incx);
subroutine vctanf(n, y, incy, x, incx);

subroutine vctan_64 (n, y, incy, x, incx);
subroutine vctanf_64(n, y, incy, x, 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.

Copyright © Adelsbach

19.4 vcasin - Vector complex arcsine $\sin^{-1}(\mathbf{x})$

```
subroutine vcasin (n, y, incy, x, incx);
subroutine vcasinf(n, y, incy, x, incx);

subroutine vcasin_64 (n, y, incy, x, incx);
subroutine vcasinf_64(n, y, incy, x, incx);
```

Given a complex input vector **x** and a result vector **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 **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.

Copyright © Adelsbach

19.5 vcacos - Vector complex arccosine $\cos^{-1}(\mathbf{x})$

```
subroutine vcacos (n, y, incy, x, incx);
subroutine vcacosf(n, y, incy, x, incx);

subroutine vcacos_64 (n, y, incy, x, incx);
subroutine vcacosf_64(n, y, incy, x, incx);
```

Given a complex input vector **x** and a result vector **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 **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.

Copyright © Adelsbach

19.6 vcatan - Vector complex arctangent $\tan^{-1}(\mathbf{x})$

```
subroutine vcatan (n, y, incy, x, incx);
subroutine vcatanf(n, y, incy, x, incx);

subroutine vcatan_64 (n, y, incy, x, incx);
subroutine vcatanf_64(n, y, incy, x, incx);
```

Given a complex input vector **x** and a result vector **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 **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.

Copyright © Adelsbach

20 Complex Hypergeometric Functions

20.1 `vcsinh` - Vector complex hyperbolic sine sinh(**x**)

```
subroutine vcsinh (n, y, incy, x, incx);
subroutine vcsinhf(n, y, incy, x, incx);

subroutine vcsinh_64 (n, y, incy, x, incx);
subroutine vcsinhf_64(n, y, incy, x, 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.

Copyright © Adelsbach

20.2 vccosh - Vector complex hyperbolic cosine cosh(**x**)

```
subroutine vcosh (n, y, incy, x, incx);
subroutine vcoshf(n, y, incy, x, incx);

subroutine vcosh_64 (n, y, incy, x, incx);
subroutine vcoshf_64(n, y, incy, x, 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.

Copyright © Adelsbach

20.3 vctanh - Vector complex hyperbolic tangent tanh(**x**)

```
subroutine vctanh (n, y, incy, x, incx);
subroutine vctanhf(n, y, incy, x, incx);

subroutine vctanh_64 (n, y, incy, x, incx);
subroutine vctanhf_64(n, y, incy, x, 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.

Copyright © Adelsbach

20.4 vcasinh - Vector complex hyperbolic arcsine $\sinh^{-1}(\mathbf{x})$

```
subroutine vcasinh (n, y, incy, x, incx);
subroutine vcasinhf(n, y, incy, x, incx);

subroutine vcasinh_64 (n, y, incy, x, incx);
subroutine vcasinhf_64(n, y, incy, x, 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.

Copyright © Adelsbach

20.5 vcacosh - Vector complex hyperbolic arccosine $\cosh^{-1}(\mathbf{x})$

```
subroutine vcacosh (n, y, incy, x, incx);
subroutine vcacoshf(n, y, incy, x, incx);

subroutine vcacosh_64 (n, y, incy, x, incx);
subroutine vcacoshf_64(n, y, incy, x, incx);
```

Given a complex input vector **x** and a result vector **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 **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.

Copyright © Adelsbach

20.6 vcatanh - Vector complex hyperbolic arctangent $\tanh^{-1}(\mathbf{x})$

```
subroutine vcatanh (n, y, incy, x, incx);
subroutine vcatanhf(n, y, incy, x, incx);

subroutine vcatanh_64 (n, y, incy, x, incx);
subroutine vcatanhf_64(n, y, incy, x, incx);
```

Given a complex input vector **x** and a result vector **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 **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.

Copyright © Adelsbach

21 Complex Arithmetic

21.1 `vcadd` - Vector complex addition $\mathbf{x} + \mathbf{y}$

```
subroutine vcadd (n, z, incz, x, incx, y, incy);
subroutine vcaddf(n, z, incz, x, incx, y, incy);

subroutine vcadd_64 (n, z, incz, x, incx, y, incy);
subroutine vcaddf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

21.2 vcsadd - Vector complex scalar subtraction $\mathbf{x}+\alpha$

```
subroutine vcsadd (n, y, incy, x, incx, a);
subroutine vcsaddf(n, y, incy, x, incx, a);

subroutine vcsadd_64 (n, y, incy, x, incx, a);
subroutine vcsaddf_64(n, y, incy, x, incx, a);
```

Given an input vector $\mathbf{x}$ and a scalar constant α 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 α.

Copyright © Adelsbach

21.3 vcsub - Vector complex subtraction $\mathbf{x} - \mathbf{y}$

```
subroutine vcsub (n, z, incz, x, incx, y, incy);
subroutine vcsubf(n, z, incz, x, incx, y, incy);

subroutine vcsub_64 (n, z, incz, x, incx, y, incy);
subroutine vcsubf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

21.4 vcssub - Vector complex scalar subtraction $\mathbf{x} - \alpha$

```
subroutine vcssub (n, y, incy, x, incx, a);
subroutine vcssubf(n, y, incy, x, incx, a);

subroutine vcssub_64 (n, y, incy, x, incx, a);
subroutine vcssubf_64(n, y, incy, x, incx, a);
```

Given an input vector $\mathbf{x}$ and a scalar constant α 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 α.

Copyright © Adelsbach

21.5 vcmul - Vector complex multiplication xy

```
subroutine vcmul (n, z, incz, x, incx, y, incy);
subroutine vcmulf(n, z, incz, x, incx, y, incy);

subroutine vcmul_64 (n, z, incz, x, incx, y, incy);
subroutine vcmulf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

21.6 vcsmul - Vector complex scalar multiplication $\alpha\mathbf{x}$

```
subroutine vcsmul (n, y, incy, x, incx, a);
subroutine vcsmulf(n, y, incy, x, incx, a);

subroutine vcsmul_64 (n, y, incy, x, incx, a);
subroutine vcsmulf_64(n, y, incy, x, incx, a);
```

Given an input vector **x** and a scalar constant α 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 α.

Copyright © Adelsbach

21.7 vcmulc - Vector complex conjugate multiplication $\mathbf{x}\bar{\mathbf{y}}$

```
subroutine vcmulc (n, z, incz, x, incx, y, incy);
subroutine vcmulcf(n, z, incz, x, incx, y, incy);

subroutine vcmulc_64 (n, z, incz, x, incx, y, incy);
subroutine vcmulcf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

21.8 vcdiv - Vector complex division x/y

```
subroutine vcdiv (n, z, incz, x, incx, y, incy);
subroutine vcdivf(n, z, incz, x, incx, y, incy);

subroutine vcdiv_64 (n, z, incz, x, incx, y, incy);
subroutine vcdivf_64(n, z, incz, x, incx, y, 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.

Copyright © Adelsbach

21.9 vcsdiv - Vector complex scalar division **x**/α

```
subroutine vcsdiv (n, y, incy, x, incx, a);
subroutine vcsdivf(n, y, incy, x, incx, a);

subroutine vcsdiv_64 (n, y, incy, x, incx, a);
subroutine vcsdivf_64(n, y, incy, x, incx, a);
```

Given an input vector **x** and a scalar constant α 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 α.

Copyright © Adelsbach

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:
```

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
