Math Vector Library 1.1 Machine Learning Extension Reference Manual (C/C++) DD-00004-011

Jan Adelsbach

February 17, 2025

Contents

1 About this Guide

1.1 Legal Information

1.2 Feedback and Contact

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

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

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

1.3 Introduction

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

1.4 Audience for This Guide

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

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

1.5 How to Use This Guide

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

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

1.6 Conventions Used in This Guide

x
: Normal math typesetting represents a normal variable.

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

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

2 Overview

2.1 Introduction

The *Math Vector Library Machine Learning Extension* is an extension to the *Math Vector Library* which provides high-performance function library with vectorized versions of standard mathematical functions. The *Machine Learning* extension builds upon the infrastructure of the *Math Vector Library* to provide performance tuned primitive functions for machine learning applications.

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 machine learning extension 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.

For the C/C++ programming language families, the 64bit variant declarations are contained in the header file **mvecml64.h**.

3 Utility

3.1 `mvecmlver` - Version query

```
# include <mvecml.h>

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

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

3.1.1 Parameters

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

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

4 Activation functions

4.1 vmlsigm - Sigmoid activation

```
# include <mvecml.h>

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

void vmlsigm_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlsigmf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the Sigmoid activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{1}{1+e^{-\mathbf{x}}}$$

4.1.1 Parameters

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

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

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

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

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

4.2 vmlsigmd - Sigmoid activation (first derivative)

```
# include <mvecml.h>

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

void vmlsigmd_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlsigmdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the Sigmoid activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{1}{1+e^{-\mathbf{x}}}\left(1-\frac{1}{1+e^{-\mathbf{x}}}\right)$$

4.2.1 Parameters

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

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

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

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

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

4.3 vmllsigm - Log-Sigmoid activation

```
# include <mvecml.h>

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

void vmllsigm_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmllsigmf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the Sigmoid activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \log\left(\frac{1}{1+e^{-\mathbf{x}}}\right)$$

4.3.1 Parameters

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

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

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

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

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

4.4 vmllsigmd - Log-Sigmoid activation (first derivative)

```
# include <mvecml.h>

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

void vmllsigmd_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmllsigmdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the Sigmoid activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{e^{-\mathbf{x}}}{e^{-\mathbf{x}}+1}$$

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.

4.5 vmlrelu - ReLU activation

```
# include <mvecml.h>

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

void vmlrelu_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlreluf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the ReLU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \begin{cases} 0 & \text{where } \mathbf{x} \leq 0 \\ x & \text{where } \mathbf{x} > 0 \end{cases}$$

4.5.1 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.

4.6 vmlrelud - ReLU activation (first derivative)

```
# include <mvecml.h>

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

void vmlrelud_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlreludf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the ReLU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \begin{cases} 0 & \text{where } \mathbf{x} < 0 \vee \mathbf{x} = 0 \\ 1 & \text{where } \mathbf{x} > 0 \end{cases}$$

The case where $\mathbf{x} = 0$ is normally undefined, for proper behavior in the application scenario of machine learning this function substitutes 0 as a result on those elements to handle this condition gracefully.

4.6.1 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.

4.7 vmltanh - Tanh activation

```
# include <mvecml.h>

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

void vmltanh_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmltanhf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the TanH activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}}$$

4.7.1 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.

4.8 vmltanhd - Tanh activation (first derivative)

```
# include <mvecml.h>

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

void vmltanhd_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmltanhdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the TanH activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq 1 - \left(\frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}} \right)^2$$

4.8.1 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.

4.9 vmltanhs - Tanhshrink activation

```
# include <mvecml.h>

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

void vmltanhs_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmltanhsf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the Tanhshrink activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \mathbf{x} - \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}}$$

4.9.1 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.

4.10 vmltanhsd - Tanhshrink activation (first derivative)

```
# include <mvecml.h>

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

void vmltanhsd_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmltanhsdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the Tanhshrink activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq 1 - \left(\frac{1}{\cosh(\mathbf{x})}\right)^2$$

4.10.1 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.

4.11 vmlsplu - Softplus activation

```
# include <mvecml.h>

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

void vmlsplu_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlspluf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the Softplus activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \ln(1 + e^{\mathbf{x}})$$

4.11.1 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.

4.12 vmlsplud - Softplus activation (first derivative)

```
# include <mvecml.h>

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

void vmlsplud_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlspludf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the Softplus activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{1}{1+e^{-\mathbf{x}}}$$

4.12.1 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.

4.13 vmlgauss - Gaussian activation

```
# include <mvecml.h>

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

void vmlgauss_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlgaussf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the Gaussian activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq e^{-\mathbf{x}^2}$$

4.13.1 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.

4.14 vmlgaussd - Gaussian activation (first derivative)

```
# include <mvecml.h>

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

void vmlgaussd_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlgaussdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the Gaussian activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq -2\mathbf{x}e^{-\mathbf{x}^2}$$

4.14.1 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.

4.15 vmlgelu - GELU activation

```
# include <mvecml.h>

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

void vmlgelu_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlgeluf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the GELU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{1}{2}\mathbf{x}\left(1+\operatorname{erf}\left(\frac{\mathbf{x}}{\sqrt{2}}\right)\right)$$

4.15.1 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.

4.16 vmlgelud - GELU activation (first derivative)

```
# include <mvecml.h>

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

void vmlgelud_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlgeludf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the GELU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{1}{2}\mathbf{x}\left(1+\operatorname{erf}\left(\frac{\mathbf{x}}{\sqrt{2}}\right)\right)+\frac{\mathbf{x}}{\sqrt{2}\sqrt{\pi}}e^{-\mathbf{x}^2/2}$$

4.16.1 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.

4.17 vmlsilu - SiLU activation

```
# include <mvecml.h>

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

void vmlsilu_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlsiluf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the SiLU activation function of the values in the **x** vector and stores the result in the **y** vector.

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

4.17.1 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.

4.18 vmlsilud - SiLU activation (first derivative)

```
# include <mvecml.h>

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

void vmlsilud_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlsiludf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the SiLU activation function of the values in the **x** vector and stores the result in the **y** vector.

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

4.18.1 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.

4.19 vmlssgn - Softsign activation

```
# include <mvecml.h>

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

void vmlssgn_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlssgnf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the Softsign activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{\mathbf{x}}{|\mathbf{x}| + 1}$$

4.19.1 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.

4.20 vmlssgnd - Softsign activation (first derivative)

```
# include <mvecml.h>

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

void vmlssgnd_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlssgndf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the Softsign activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{|\mathbf{x}|}{(\mathbf{x}^2+1)|\mathbf{x}|+2\mathbf{x}^2}$$

4.20.1 Parameters

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

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

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

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

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

4.21 vmlmish - MISH activation

```
# include <mvecml.h>

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

void vmlmish_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlmishf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the MISH activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \tanh(\log(e^{\mathbf{x}} + 1))\mathbf{x}$$

4.21.1 Parameters

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

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

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

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

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

4.22 vmlmishd - MISH activation (first derivative)

```
# include <mvecml.h>

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

void vmlmishd_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlmishdf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the MISH activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \tanh(\log(e^{\mathbf{x}}+1)) + \frac{\mathbf{x}e^{\mathbf{x}}\left(\frac{1}{\cosh(\log(e^{\mathbf{x}}+1))}\right)^2}{e^{\mathbf{x}}+1}$$

4.22.1 Parameters

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

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

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

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

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

4.23 vmlatan - Arctan activation

```
# include <mvecml.h>

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

void vmlatan_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlatanf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the arc-tangent activation function of the values in the **x** vector and stores the result in the **y** vector.

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

4.23.1 Parameters

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

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

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

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

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

4.24 vmlatand - Arctan activation (first derivative)

```
# include <mvecml.h>

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

void vmlatand_64(long n, double *restrict y, long incy, const double *restrict x, long incx);
void vmlatandf_64(long n, float *restrict y, long incy, const float *restrict x, long incx);
```

Given an input vector **x** and a result vector **y** this function computes the first derivative of the arc-tangent activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{1}{\mathbf{x}^2 + 1}$$

4.24.1 Parameters

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

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

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

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

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

4.25 vmlprelu - PReLU activation

```
# include <mvecml.h>

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

void vmlprelu_64(long n, double *restrict y, long incy,
                 double a, const double *restrict x, long incx);
void vmlpreluf_64(long n, float *restrict y, long incy,
                  float a, const float *restrict x, long incx);
```

Given an input vector **x**, a result vector **y** and a constant α this function computes the PReLU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \begin{cases} \alpha\mathbf{x} & \text{where } \mathbf{x} < 0 \\ \mathbf{x} & \text{where } \mathbf{x} \geq 0 \end{cases}$$

4.25.1 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.

`A` - **REAL** *ENTRY:* Constant α.

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

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

4.26 vmlprelud - PReLU activation (first derivative)

```
# include <mvecml.h>

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

void vmlprelud_64(long n, double *restrict y, long incy, double a,
                  const double *restrict x, long incx);
void vmlpreludf_64(long n, float *restrict y, long incy, float a,
                   const float *restrict x, long incx);
```

Given an input vector **x**, a result vector **y** and a constant α this function computes the first derivative of the PReLU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \begin{cases} \alpha & \text{where } \mathbf{x} < 0 \\ 1 & \text{where } \mathbf{x} \geq 0 \end{cases}$$

4.26.1 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.

`A` - **REAL** *ENTRY:* Constant α.

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

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

4.27 vmlselu - SELU activation

```
# include <mvecml.h>

void vmlselu (int n, double *restrict y, int incy,
              double l, double a, const double *restrict x, int incx);
void vmlseluf(int n, float *restrict y, int incy,
              float l, float a, const float *restrict x, int incx);

void vmlselu_64(long n, double *restrict y, long incy,
                double l, double a, const double *restrict x, long incx);
void vmlseluf_64(long n, float *restrict y, long incy,
                 float l, float a, const float *restrict x, long incx);
```

Given an input vector **x**, a result vector **y** and constants λ and α this function computes the SELU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \lambda \begin{cases} \alpha(e^{\mathbf{x}} - 1) & \text{where } \mathbf{x} < 0 \\ \mathbf{x} & \text{where } \mathbf{x} \geq 0 \end{cases}$$

4.27.1 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.

`L` - **REAL** *ENTRY:* Constant λ.

`A` - **REAL** *ENTRY:* Constant α.

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

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

4.28 vmlselud - SELU activation (first derivative)

```
# include <mvecml.h>

void vmlselud (int n, double *restrict y, int incy,
               double l, double a, const double *restrict x, int incx);
void vmlseludf(int n, float *restrict y, int incy,
               float l, float a, const float *restrict x, int incx);

void vmlselud_64(long n, double *restrict y, long incy,
                 double l, double a, const double *restrict x, long incx);
void vmlseludf_64(long n, float *restrict y, long incy,
                  float l, float a, const float *restrict x, long incx);
```

Given an input vector **x**, a result vector **y** and constants λ and α this function computes the first derivative of the SELU activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \lambda \begin{cases} \alpha e^{\mathbf{x}} & \text{where } \mathbf{x} < 0 \\ 1 & \text{where } \mathbf{x} \geq 0 \end{cases}$$

4.28.1 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.

`L` - **REAL** *ENTRY:* Constant λ.

`A` - **REAL** *ENTRY:* Constant α.

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

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

4.29 vmlsmht - SMHT activation

```
# include <mvecml.h>

void vmlsmht(int n, double *restrict y, int incy,
             double a, double b, double c, double d, const double *restrict x, int incx);
void vmlsmhtf(int n, float *restrict y, int incy,
              float a, float b, float c, float d, const float *restrict x, int incx);

void vmlsmht_64(long n, double *restrict y, long incy,
                double a, double b, double c, double d, const double *restrict x, long incx);
void vmlsmhtf_64(long n, float *restrict y, long incy,
                 float a, float b, float c, float d, const float *restrict x, long incx);
```

Given an input vector **x**, a result vector **y** and constants λ and α this function computes the *Soboleva modified hyperbolic tangent* (SMHT) activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{e^{a\mathbf{x}} - e^{-b\mathbf{x}}}{e^{c\mathbf{x}} + e^{-d\mathbf{x}}}$$

4.29.1 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.

A - **REAL** *ENTRY:* Constant a.

B - **REAL** *ENTRY:* Constant b.

C - **REAL** *ENTRY:* Constant c.

D - **REAL** *ENTRY:* Constant d.

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

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

4.30 vmlsmhtd - SMHT activation (first derivative)

```
# include <mvecml.h>

void vmlsmhtd(int n, double *restrict y, int incy,
              double a, double b, double c, double d, const double *restrict x, int incx);
void vmlsmhtdf(int n, float *restrict y, int incy,
               float a, float b, float c, float d, const float *restrict x, int incx);

void vmlsmhtd_64(long n, double *restrict y, long incy,
                 double a, double b, double c, double d, const double *restrict x, long incx);
void vmlsmhtdf_64(long n, float *restrict y, long incy,
                  float a, float b, float c, float d, const float *restrict x, long incx);
```

Given an input vector **x**, a result vector **y** and constants λ and α this function computes the first derivative of the *Soboleva modified hyperbolic tangent* (SMHT) activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \frac{ae^{a\mathbf{x}} + be^{-b\mathbf{x}}}{e^{c\mathbf{x}} + e^{-d\mathbf{x}}} - \mathrm{smht}(\mathbf{x})\frac{\mathrm{ce}^{c\mathbf{x}} - \mathrm{de}^{-d\mathbf{x}}}{\mathrm{e}^{c\mathbf{x}} + \mathrm{e}^{-d\mathbf{x}}}$$

4.30.1 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.

`A` - **REAL** *ENTRY:* Constant a.

`B` - **REAL** *ENTRY:* Constant b.

`C` - **REAL** *ENTRY:* Constant c.

`D` - **REAL** *ENTRY:* Constant d.

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

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

4.31 vmlelud - ELU activation

```
# include <mvecml.h>

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

void vmlelu_64(long n, double *restrict y, long incy,
               double a, const double *restrict x, long incx);
void vmleluf_64(long n, float *restrict y, long incy,
                float a, const float *restrict x, long incx);
```

Given an input vector **x**, a result vector **y** and constants λ and α this function computes the *Exponential Linear Unit* (ELU) activation function of the values in the **x** vector and stores the result in the **y** vector.

$$\mathbf{y} \doteq \begin{cases} \mathbf{x} & \text{where } \mathbf{x} > 0 \\ \alpha(\exp(\mathbf{x}) - 1)) & \text{where } \mathbf{x} \leq 0 \end{cases}$$

4.31.1 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.

`A` - **REAL** *ENTRY:* Constant a.

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

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

4.32 vmlelud - ELU activation (first derivative)

```
# include <mvecml.h>

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

void vmlelud_64(long n, double *restrict y, long incy,
                double a, const double *restrict x, long incx);
void vmleludf_64(long n, float *restrict y, long incy,
                 float a, const float *restrict x, long incx);
```

Given an input vector $\mathbf{x}$, a result vector $\mathbf{y}$ and constants λ and α this function computes the first derivative of the *Exponential Linear Unit* (ELU) activation function of the values in the $\mathbf{x}$ vector and stores the result in the $\mathbf{y}$ vector.

$$\mathbf{y} \doteq \begin{cases} 1 & \text{where } \mathbf{x} > 0 \\ \alpha \exp(\mathbf{x}) & \text{where } \mathbf{x} \leq 0 \end{cases}$$

4.32.1 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.

`A` - **REAL** *ENTRY:* Constant a.

`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.

4.33 vmlcelu - CELU activation

```
# include <mvecml.h>

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

void vmlcelu_64(long n, double *restrict y, long incy,
                double a, const double *restrict x, long incx);
void vmlceluf_64(long n, float *restrict y, long incy,
                 float a, const float *restrict x, long incx);
```

Given an input vector $\mathbf{x}$, a result vector $\mathbf{y}$ and constants λ and α this function computes the *Continuously Differentiable Exponential Linear Unit* (CELU) activation function of the values in the $\mathbf{x}$ vector and stores the result in the $\mathbf{y}$ vector.

$$\mathbf{y} \doteq \begin{cases} \mathbf{x} & \text{where } \mathbf{x} > 0 \\ \alpha\left(\exp\left(\frac{\mathbf{x}}{\alpha}\right) - 1\right) & \text{where } \mathbf{x} \leq 0 \end{cases}$$

4.33.1 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.

`A` - **REAL** *ENTRY:* Constant a.

`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.

4.34 vmlcelud - CELU activation (first derivative)

```
# include <mvecml.h>

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

void vmlcelud_64(long n, double *restrict y, long incy,
                 double a, const double *restrict x, long incx);
void vmlceludf_64(long n, float *restrict y, long incy,
                  float a, const float *restrict x, long incx);
```

Given an input vector $\mathbf{x}$, a result vector $\mathbf{y}$ and constants λ and α this function computes the first derivative of the *Continuously Differentiable Exponential Linear Unit* (CELU) activation function of the values in the $\mathbf{x}$ vector and stores the result in the $\mathbf{y}$ vector.

$$\mathbf{y} \doteq \begin{cases} 1 & \text{where } \mathbf{x} > 0 \\ \exp\left(\frac{\mathbf{x}}{\alpha}\right) & \text{where } \mathbf{x} \leq 0 \end{cases}$$

4.34.1 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.

`A` - **REAL** *ENTRY:* Constant α.

`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.

5 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:
```