
vDSP Single-Vector Operations Reference

[Performance > Carbon](#)



2009-01-07



Apple Inc.
© 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Carbon, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY

DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

vDSP Single-Vector Operations Reference 7

Overview	7
Functions by Task	7
Finding Absolute Values of Elements	7
Negating Values of Elements	7
Filling or Clearing Elements	8
Building a Vector With Generated Values	8
Squaring Element Values	9
Converting Between Polar and Rectangular Coordinates	9
Converting to Decibel Equivalents	9
Extracting Fractional Parts of Elements	9
Conjugating Complex Vectors	9
Squaring Magnitudes of Complex Elements	10
Calculating Phase Values of Complex Elements	10
Clipping, Limit, and Threshold Operations	10
Compressing Element Values	11
Gathering Elements of a Vector	11
Using Indices to Select Elements of a Vector	11
Reversing the Order of Elements	11
Copying Complex Vectors	12
Finding Zero Crossings	12
Calculating Linear Average of a Vector	12
Performing Linear Interpolation Between Neighboring Elements	12
Integrating a Vector	12
Sorting a Vector	13
Calculating Sliding-Window Sums	13
Converting Between Single and Double Precision	13
Functions	13
vDSP_nzcros	13
vDSP_nzcrosD	14
vDSP_polar	16
vDSP_polarD	16
vDSP_rect	17
vDSP_rectD	18
vDSP_vabs	19
vDSP_vabsD	20
vDSP_vabsi	20
vDSP_vavlin	21
vDSP_vavlinD	22
vDSP_vclip	23
vDSP_vclipc	24

vDSP_vclipC	25
vDSP_vclipD	26
vDSP_vclr	27
vDSP_vclrD	27
vDSP_vcmprs	27
vDSP_vcmprsD	28
vDSP_vdbcon	29
vDSP_vdbconD	30
vDSP_vdpsp	31
vDSP_vfill	32
vDSP_vfillD	33
vDSP_vfilli	33
vDSP_vfrac	34
vDSP_vfracD	35
vDSP_vgathr	36
vDSP_vgathra	36
vDSP_vgathraD	37
vDSP_vgathrD	38
vDSP_vgen	39
vDSP_vgenD	39
vDSP_vgenp	40
vDSP_vgenpD	42
vDSP_viclip	43
vDSP_viclipD	44
vDSP_vindex	45
vDSP_vindexD	46
vDSP_vlim	47
vDSP_vlimD	47
vDSP_vlint	48
vDSP_vlintD	49
vDSP_vnabs	51
vDSP_vnabsD	51
vDSP_vneg	52
vDSP_vnegD	53
vDSP_vramp	53
vDSP_vrampD	54
vDSP_vrsum	55
vDSP_vrsumD	55
vDSP_vrvrs	56
vDSP_vrvrsD	57
vDSP_vsimps	57
vDSP_vsimpsD	58
vDSP_vsort	59
vDSP_vsortD	60
vDSP_vsorti	60
vDSP_vsortiD	61

vDSP_vspdp 62
vDSP_vsq 62
vDSP_vsqD 63
vDSP_vssq 63
vDSP_vssqD 64
vDSP_vswsum 64
vDSP_vswsumD 65
vDSP_vtabi 66
vDSP_vtabiD 67
vDSP_vthr 68
vDSP_vthrD 69
vDSP_vthres 70
vDSP_vthresD 70
vDSP_vthrsc 71
vDSP_vthrscD 72
vDSP_vtrapz 73
vDSP_vtrapzD 74
vDSP_zvabs 75
vDSP_zvabsD 75
vDSP_zvconj 76
vDSP_zvconjD 77
vDSP_zvfill 77
vDSP_zvfillD 78
vDSP_zvmags 79
vDSP_zvmagsD 79
vDSP_zvmgsa 80
vDSP_zvmgsaD 81
vDSP_zvmov 82
vDSP_zvmovD 82
vDSP_zvneg 83
vDSP_zvnegD 84
vDSP_zvphas 85
vDSP_zvphasD 85

Document Revision History 87

Index 89

vDSP Single-Vector Operations Reference

Framework:	Accelerate/vecLib
Declared in	vDSP.h

Overview

This document describes the C API for performing common routines on a single vector in vDSP. These functions perform tasks such as finding the absolute value of a vector, compressing the values of a vector, or converting between single and double precision vectors.

Functions by Task

Finding Absolute Values of Elements

- [vDSP_vabs](#) (page 19)
Vector absolute values; single precision.
- [vDSP_vabsD](#) (page 20)
Vector absolute values; double precision.
- [vDSP_vabsi](#) (page 20)
Integer vector absolute values.
- [vDSP_zvabs](#) (page 75)
Complex vector absolute values; single precision.
- [vDSP_zvabsD](#) (page 75)
Complex vector absolute values; double precision.
- [vDSP_vnabs](#) (page 51)
Vector negative absolute values; single precision.
- [vDSP_vnabsD](#) (page 51)
Vector negative absolute values; double precision.

Negating Values of Elements

- [vDSP_vneg](#) (page 52)
Vector negative values; single precision.
- [vDSP_vnegD](#) (page 53)
Vector negative values; double precision.

- [vDSP_zvneg](#) (page 83)
Complex vector negate; single precision.
- [vDSP_zvnegD](#) (page 84)
Complex vector negate; double precision.

Filling or Clearing Elements

- [vDSP_vfill](#) (page 32)
Vector fill; single precision.
- [vDSP_vfillD](#) (page 33)
Vector fill; double precision.
- [vDSP_vfilli](#) (page 33)
Integer vector fill.
- [vDSP_zvfill](#) (page 77)
Complex vector fill; single precision.
- [vDSP_zvfillD](#) (page 78)
Complex vector fill; double precision.
- [vDSP_vclr](#) (page 27)
Vector clear; single precision.
- [vDSP_vclrD](#) (page 27)
Vector clear; double precision.

Building a Vector With Generated Values

- [vDSP_vramp](#) (page 53)
Build ramped vector; single precision.
- [vDSP_vrampD](#) (page 54)
Build ramped vector; double precision.
- [vDSP_vgen](#) (page 39)
Vector tapered ramp; single precision.
- [vDSP_vgenD](#) (page 39)
Vector tapered ramp; double precision.
- [vDSP_vgenp](#) (page 40)
Vector generate by extrapolation and interpolation; single precision.
- [vDSP_vgenpD](#) (page 42)
Vector generate by extrapolation and interpolation; double precision.
- [vDSP_vtabi](#) (page 66)
Vector interpolation, table lookup; single precision.
- [vDSP_vtabiD](#) (page 67)
Vector interpolation, table lookup; double precision.

Squaring Element Values

[vDSP_vsq](#) (page 62)

Computes the squared values of vector `input` and leaves the result in vector `result`; single precision.

[vDSP_vsqD](#) (page 63)

Computes the squared values of vector `signal1` and leaves the result in vector `result`; double precision.

[vDSP_vssq](#) (page 63)

Computes the signed squares of vector `signal1` and leaves the result in vector `result`; single precision.

[vDSP_vssqD](#) (page 64)

Computes the signed squares of vector `signal1` and leaves the result in vector `result`; double precision.

Converting Between Polar and Rectangular Coordinates

[vDSP_polar](#) (page 16)

Rectangular to polar conversion; single precision.

[vDSP_polarD](#) (page 16)

Rectangular to polar conversion; double precision.

[vDSP_rect](#) (page 17)

Polar to rectangular conversion; single precision.

[vDSP_rectD](#) (page 18)

Polar to rectangular conversion; double precision.

Converting to Decibel Equivalents

[vDSP_vdbcon](#) (page 29)

Vector convert power or amplitude to decibels; single precision.

[vDSP_vdbconD](#) (page 30)

Vector convert power or amplitude to decibels; double precision.

Extracting Fractional Parts of Elements

[vDSP_vfrac](#) (page 34)

Vector truncate to fraction; single precision.

[vDSP_vfracD](#) (page 35)

Vector truncate to fraction; double precision.

Conjugating Complex Vectors

[vDSP_zvconj](#) (page 76)

Complex vector conjugate; single precision.

[vDSP_zvconjD](#) (page 77)
Complex vector conjugate; double precision.

Squaring Magnitudes of Complex Elements

[vDSP_zvmags](#) (page 79)
Complex vector magnitudes squared; single precision.

[vDSP_zvmagsD](#) (page 79)
Complex vector magnitudes squared; double precision.

[vDSP_zvmgsa](#) (page 80)
Complex vector magnitudes square and add; single precision.

[vDSP_zvmgsaD](#) (page 81)
Complex vector magnitudes square and add; double precision.

Calculating Phase Values of Complex Elements

[vDSP_zvphas](#) (page 85)
Complex vector phase; single precision.

[vDSP_zvphasD](#) (page 85)
Complex vector phase; double precision.

Clipping, Limit, and Threshold Operations

[vDSP_vclip](#) (page 23)
Vector clip; single precision.

[vDSP_vclipD](#) (page 26)
Vector clip; double precision.

[vDSP_vclipc](#) (page 24)
Vector clip and count; single precision.

[vDSP_vclipcD](#) (page 25)
Vector clip and count; double precision.

[vDSP_viclip](#) (page 43)
Vector inverted clip; single precision.

[vDSP_viclipD](#) (page 44)
Vector inverted clip; double precision.

[vDSP_vlim](#) (page 47)
Vector test limit; single precision.

[vDSP_vlimD](#) (page 47)
Vector test limit; double precision.

[vDSP_vthr](#) (page 68)
Vector threshold; single precision.

[vDSP_vthrD](#) (page 69)
Vector threshold; double precision.

[vDSP_vthres](#) (page 70)

Vector threshold with zero fill; single precision.

[vDSP_vthresD](#) (page 70)

Vector threshold with zero fill; double precision.

[vDSP_vthrsc](#) (page 71)

Vector threshold with signed constant; single precision.

[vDSP_vthrscD](#) (page 72)

Vector threshold with signed constant; double precision.

Compressing Element Values

[vDSP_vcmpsr](#) (page 27)

Vector compress; single precision.

[vDSP_vcmpsrD](#) (page 28)

Vector compress; double precision.

Gathering Elements of a Vector

[vDSP_vgather](#) (page 36)

Vector gather; single precision.

[vDSP_vgatherD](#) (page 38)

Vector gather; double precision.

[vDSP_vgathera](#) (page 36)

Vector gather, absolute pointers; single precision.

[vDSP_vgatheraD](#) (page 37)

Vector gather, absolute pointers; double precision.

Using Indices to Select Elements of a Vector

[vDSP_vindex](#) (page 45)

Vector index; single precision.

[vDSP_vindexD](#) (page 46)

Vector index; double precision.

Reversing the Order of Elements

[vDSP_vrvrs](#) (page 56)

Vector reverse order, in place; single precision.

[vDSP_vrvrsD](#) (page 57)

Vector reverse order, in place; double precision.

Copying Complex Vectors

[vDSP_zvmov](#) (page 82)
Complex vector copy; single precision.

[vDSP_zvmovD](#) (page 82)
Complex vector copy; double precision.

Finding Zero Crossings

[vDSP_nzcros](#) (page 13)
Find zero crossings; single precision.

[vDSP_nzcrosD](#) (page 14)
Find zero crossings; double precision.

Calculating Linear Average of a Vector

[vDSP_vavlin](#) (page 21)
Vector linear average; single precision.

[vDSP_vavlinD](#) (page 22)
Vector linear average; double precision.

Performing Linear Interpolation Between Neighboring Elements

[vDSP_vlint](#) (page 48)
Vector linear interpolation between neighboring elements; single precision.

[vDSP_vlintD](#) (page 49)
Vector linear interpolation between neighboring elements; double precision.

Integrating a Vector

[vDSP_vrsum](#) (page 55)
Vector running sum integration; single precision.

[vDSP_vrsumD](#) (page 55)
Vector running sum integration; double precision.

[vDSP_vsimps](#) (page 57)
Simpson integration; single precision.

[vDSP_vsimpsD](#) (page 58)
Simpson integration; double precision.

[vDSP_vtrapz](#) (page 73)
Vector trapezoidal integration; single precision.

[vDSP_vtrapzD](#) (page 74)
Vector trapezoidal integration; double precision.

Sorting a Vector

[vDSP_vsort](#) (page 59)

Vector in-place sort; single precision.

[vDSP_vsortD](#) (page 60)

Vector in-place sort; double precision.

[vDSP_vsorti](#) (page 60)

Vector index in-place sort; single precision.

[vDSP_vsortiD](#) (page 61)

Vector index in-place sort; double precision.

Calculating Sliding-Window Sums

[vDSP_vswsum](#) (page 64)

Vector sliding window sum; single precision.

[vDSP_vswsumD](#) (page 65)

Vector sliding window sum; double precision.

Converting Between Single and Double Precision

[vDSP_vdpsp](#) (page 31)

Vector convert double-precision to single-precision.

[vDSP_vspdp](#) (page 62)

Vector convert single-precision to double-precision.

Functions

vDSP_nzcros

Find zero crossings; single precision.

```
void vDSP_nzcros (float * A,
                 vDSP_Stride I,
                 vDSP_Length B,
                 vDSP_Length * C,
                 vDSP_Length * D,
                 vDSP_Length N);
```

Parameters

A

Single-precision real input vector

I

Stride for A

<i>B</i>	Maximum number of crossings to find
<i>C</i>	Index of last crossing found
<i>D</i>	Total number of zero crossings found
<i>N</i>	Count of elements in A

Discussion

This performs the operation

$$d = c = 0$$

For integer n , $0 < n < N$;

If $\text{sign}(A_{nI}) \neq \text{sign}(A_{(n-1)I})$ then

$$d = d + 1$$

If $d = B$ then

$$c = nI,$$

exit. $n = \{1, N-1\}$

The "function" $\text{sign}(x)$ above has the value -1 if the sign bit of x is 1 (x is negative or -0), and +1 if the sign bit is 0 (x is positive or +0).

Scans vector A to locate transitions from positive to negative values and from negative to positive values. The scan terminates when the number of crossings specified by B is found, or the end of the vector is reached. The zero-based index of the last crossing is returned in C. C is the actual array index, not the pre-stride index. If the zero crossing that B specifies is not found, zero is returned in C. The total number of zero crossings found is returned in D.

Note that a transition from -0 to +0 or from +0 to -0 is counted as a zero crossing.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_nzcrosD

Find zero crossings; double precision.

```
void vDSP_nzcrosD (double * A,
vDSP_Stride I,
vDSP_Length B,
vDSP_Length * C,
vDSP_Length * D,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Maximum number of crossings to find
<i>C</i>	Index of last crossing found
<i>D</i>	Total number of zero crossings found
<i>N</i>	Count of elements in A

Discussion

This performs the operation

$$d = c = 0$$

For integer n , $0 < n < N$;

If $\text{sign}(A_{nI}) \neq \text{sign}(A_{(n-1)I})$ then

$$d = d + 1$$

If $d = B$ then

$$c = nI,$$

exit. $n = \{1, N-1\}$

The "function" $\text{sign}(x)$ above has the value -1 if the sign bit of x is 1 (x is negative or -0), and +1 if the sign bit is 0 (x is positive or +0).

Scans vector A to locate transitions from positive to negative values and from negative to positive values. The scan terminates when the number of crossings specified by B is found, or the end of the vector is reached. The zero-based index of the last crossing is returned in C . C is the actual array index, not the pre-stride index. If the zero crossing that B specifies is not found, zero is returned in C . The total number of zero crossings found is returned in D .

Note that a transition from -0 to +0 or from +0 to -0 is counted as a zero crossing.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_polar

Rectangular to polar conversion; single precision.

```
void vDSP_polar (float * A,
                vDSP_Stride I,
                float * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A, must be even
<i>C</i>	Single-precision output vector
<i>K</i>	Stride for C, must be even
<i>N</i>	Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = \sqrt{A_{nI}^2 + A_{nI+1}^2} \quad C_{nK+1} = \text{atan2}(A_{nI+1}, A_{nI}), \quad n = \{0, N-1\}$$

Converts rectangular coordinates to polar coordinates. Cartesian (x,y) pairs are read from vector A. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range [-pi, pi] are written to vector C. N specifies the number of coordinate pairs in A and C.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_rect`, which converts polar to rectangular coordinates.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_polarD

Rectangular to polar conversion; double precision.


```
void vDSP_polarD (double * A,
                 vDSP_Stride I,
                 double * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for <i>A</i> , must be even
<i>C</i>	Double-precision output vector
<i>K</i>	Stride for <i>C</i> , must be even
<i>N</i>	Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = \sqrt{A_{nI}^2 + A_{nI+1}^2} \quad C_{nK+1} = \text{atan2}(A_{nI+1}, A_{nI}), \quad n = \{0, N-1\}$$

Converts rectangular coordinates to polar coordinates. Cartesian (x,y) pairs are read from vector *A*. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range [-pi, pi] are written to vector *C*. *N* specifies the number of coordinate pairs in *A* and *C*.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_rectD`, which converts polar to rectangular coordinates.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_rect

Polar to rectangular conversion; single precision.

```
void vDSP_rect (float * A,
               vDSP_Stride I,
               float * C,
               vDSP_Stride K,
               vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
----------	------------------------------------

<i>I</i>	Stride for A, must be even
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C, must be even
<i>N</i>	Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = A_{nI} \cdot \cos(A_{nI+1}) \quad C_{nK+1} = A_{nI} \cdot \sin(A_{nI+1}) \quad n = \{0, N-1\}$$

Converts polar coordinates to rectangular coordinates. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range $[-\pi, \pi]$ are read from vector A. Cartesian (x,y) pairs are written to vector C. N specifies the number of coordinate pairs in A and C.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_polar`, which converts rectangular to polar coordinates.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_rectD

Polar to rectangular conversion; double precision.

```
void vDSP_rectD (double * A,
                vDSP_Stride I,
                double * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A, must be even
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C, must be even
<i>N</i>	Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = A_{nI} \cdot \cos(A_{nI+1}) \quad C_{nK+1} = A_{nI} \cdot \sin(A_{nI+1}) \quad n = \{0, N-1\}$$

Converts polar coordinates to rectangular coordinates. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range [-pi, pi] are read from vector A. Cartesian (x,y) pairs are written to vector C. N specifies the number of coordinate pairs in A and C.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_polarD`, which converts rectangular to polar coordinates.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vabs

Vector absolute values; single precision.

```
void vDSP_vabs (float * A,
               vDSP_Stride I,
               float * C,
               vDSP_Stride K,
               vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

This performs the operation

$$C_{nK} = |A_{nI}|, \quad n = \{0, N-1\}$$

Writes the absolute values of the elements of A into corresponding elements of C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vabsD

Vector absolute values; double precision.

```
void vDSP_vabsD (double * A,
                vDSP_Stride I,
                double * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters*A*

Double-precision real input vector

*I*Stride for *A**C*

Double-precision real output vector

*K*Stride for *C**N*

Count

Discussion

This performs the operation

$$C_{nK} = |A_{nI}|, \quad n = \{0, N-1\}$$

Writes the absolute values of the elements of *A* into corresponding elements of *C*.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vabsi

Integer vector absolute values.

```
void vDSP_vabsi (int * A,
                vDSP_Stride I,
                int * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters*A*

Integer input vector

I
Stride for A

C
Integer output vector

K
Stride for C

N
Count

Discussion

This performs the operation

$$C_{nK} = |A_{nI}|, \quad n = \{0, N-1\}$$

Writes the absolute values of the elements of A into corresponding elements of C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vavlin

Vector linear average; single precision.

```
void vDSP_vavlin (float * A,
                 vDSP_Stride I,
                 float * B,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

A
Single-precision real input vector

I
Stride for A

B
Single-precision real input scalar

C
Single-precision real input-output vector

K
Stride for C

N
Count ; each vector must have at least N elements

Discussion

This performs the operation

$$C_{nK} = \frac{C_{nK}B + A_{nI}}{B + 1.0}, \quad n = \{0, N-1\}$$

Recalculates the linear average of input-output vector *C* to include input vector *A*. Input scalar *B* specifies the number of vectors included in the current average.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vavlinD

Vector linear average; double precision.

```
void vDSP_vavlinD (double * A,
vDSP_Stride I,
double * B,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for <i>A</i>
<i>B</i>	Double-precision real input scalar
<i>C</i>	Double-precision real input-output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count ; each vector must have at least <i>N</i> elements

Discussion

This performs the operation

$$C_{nK} = \frac{C_{nK}B + A_{nI}}{B + 1.0}, \quad n = \{0, N-1\}$$

Recalculates the linear average of input-output vector *C* to include input vector *A*. Input scalar *B* specifies the number of vectors included in the current average.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vclip

Vector clip; single precision.

```
void vDSP_vclip (float * A,
                vDSP_Stride I,
                float * B,
                float * C,
                float * D,
                vDSP_Stride L,
                vDSP_Length N);
```

Parameters*A*

Single-precision real input vector

*I*Stride for *A**B*

Single-precision real input scalar: low clipping threshold

C

Single-precision real input scalar: high clipping threshold

D

Single-precision real output vector

*L*Stride for *D**N*

Count

Discussion

This performs the operation

$$\left\{ \begin{array}{ll} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C, \quad n = \{0, N-1\} \\ \text{If } b \leq A_{nI} \leq c & \text{then } D_{nM} = A_{nI} \end{array} \right.$$

Elements of *A* are copied to *D* while clipping elements that are outside the interval [*B*, *C*] to the endpoints.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vclipc

Vector clip and count; single precision.

```
void vDSP_vclipc (float * A,
                 vDSP_Stride I,
                 float * B,
                 float * C,
                 float * D,
                 vDSP_Stride L,
                 vDSP_Length N,
                 vDSP_Length * NLOW,
                 vDSP_Length * NHI);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Single-precision real input scalar: low clipping threshold
<i>C</i>	Single-precision real input scalar: high clipping threshold
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count of elements in A and D
<i>NLOW</i>	Number of elements that were clipped to B
<i>NHI</i>	Number of elements that were clipped to C

Discussion

This performs the operation

$$\left\{ \begin{array}{ll} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C, \quad n = \{0, N-1\} \\ \text{If } b \leq A_{nI} \leq c & \text{then } D_{nM} = A_{nI} \end{array} \right.$$

Elements of A are copied to D while clipping elements that are outside the interval [B, C] to the endpoints.

The count of elements clipped to B is returned in *NLOW, and the count of elements clipped to C is returned in *NHI

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vclipcD

Vector clip and count; double precision.

```
void vDSP_vclipcD (double * A,
vDSP_Stride I,
double * B,
double * C,
double * D,
vDSP_Stride L,
vDSP_Length N,
vDSP_Length * NLOW,
vDSP_Length * NHI);
```

Parameters*A*

Double-precision real input vector

I

Stride for A

B

Double-precision real input scalar: low clipping threshold

C

Double-precision real input scalar: high clipping threshold

D

Double-precision real output vector

L

Stride for D

N

Count of elements in A and D

NLOW

Number of elements that were clipped to B

NHI

Number of elements that were clipped to C

Discussion

This performs the operation

$$\left\{ \begin{array}{ll} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C, \quad n = \{0, N-1\} \\ \text{If } b \leq A_{nI} \leq c & \text{then } D_{nM} = A_{nI} \end{array} \right.$$

Elements of A are copied to D while clipping elements that are outside the interval [B, C] to the endpoints.

The count of elements clipped to B is returned in *NLOW, and the count of elements clipped to C is returned in *NHI

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vclipD

Vector clip; double precision.

```
void vDSP_vclipD (double * A,
                 vDSP_Stride I,
                 double * B,
                 double * C,
                 double * D,
                 vDSP_Stride L,
                 vDSP_Length N);
```

Parameters*A*

Double-precision real input vector

*I*Stride for *A**B*

Double-precision real input scalar: low clipping threshold

C

Double-precision real input scalar: high clipping threshold

D

Double-precision real output vector

*L*Stride for *D**N*

Count

Discussion

This performs the operation

$$\left\{ \begin{array}{ll} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C, \quad n = \{0, N-1\} \\ \text{If } b \leq A_{nI} \leq c & \text{then } D_{nM} = A_{nI} \end{array} \right.$$

Elements of *A* are copied to *D* while clipping elements that are outside the interval [*B*, *C*] to the endpoints.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vclr

Vector clear; single precision.

```
void vDSP_vclr (float * C,
               vDSP_Stride K,
               vDSP_Length N);
```

Parameters

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

All elements of vector *C* are set to zeros.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vclrD

Vector clear; double precision.

```
void vDSP_vclrD (double * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

All elements of vector *C* are set to zeros.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vcmprs

Vector compress; single precision.

```
void vDSP_vcmprs (float * A,
                 vDSP_Stride I,
                 float * B,
                 vDSP_Stride J,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Single-precision real input vector
<i>J</i>	Stride for B
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

$$p = 0$$

If $B_{nJ} \neq 0.0$ then $C_{pK} = A_{nI}$; $p = p + 1$; $n = \{0, N-1\}$

Compresses vector A based on the nonzero values of gating vector B. For nonzero elements of B, corresponding elements of A are sequentially copied to output vector C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vcmprsD

Vector compress; double precision.

```
void vDSP_vcmprsD (double * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input vector
<i>J</i>	Stride for B
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

$$p = 0$$

If $B_{nJ} \neq 0.0$ then $C_{pK} = A_{nI}$; $p = p + 1$; $n = \{0, N-1\}$

Compresses vector A based on the nonzero values of gating vector B. For nonzero elements of B, corresponding elements of A are sequentially copied to output vector C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdbcon

Vector convert power or amplitude to decibels; single precision.

```
void vDSP_vdbcon (float * A,
                 vDSP_Stride I,
                 float * B,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N,
                 unsigned int F);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Single-precision real input scalar: zero reference
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count
<i>F</i>	Power (0) or amplitude (1) flag

Discussion

Performs the following operation. Uses 20 if F is 1, or 10 if F is 0.

$$C_{nK} = \alpha \left(\log_{10} \left(\frac{A_{nI}}{B} \right) \right) \quad n = \{0, N-1\}$$

Converts inputs from vector A to their decibel equivalents, calculated in terms of power or amplitude according to flag F. As a relative reference point, the value of input scalar B is considered to be zero decibels.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdbconD

Vector convert power or amplitude to decibels; double precision.

```
void vDSP_vdbconD (double * A,
vDSP_Stride I,
double * B,
double * C,
vDSP_Stride K,
vDSP_Length N,
unsigned int F);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input scalar: zero reference
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count
<i>F</i>	Power (0) or amplitude (1) flag

Discussion

Performs the operation. The Greek letter alpha equals 20 if $F = 1$, and 10 if $F = 0$.

$$C_{nK} = \alpha \left(\log_{10} \left(\frac{A_{nI}}{B} \right) \right) \quad n = \{0, N-1\}$$

Converts inputs from vector *A* to their decibel equivalents, calculated in terms of power or amplitude according to flag *F*. As a relative reference point, the value of input scalar *B* is considered to be zero decibels.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdpsp

Vector convert double-precision to single-precision.

```
void vDSP_vdpsp (double * A,
vDSP_Stride I,
float * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

This performs the operation

$$C_{nK} = A_{nI}, \quad n = \{0, N-1\}$$

Creates single-precision vector *C* by converting double-precision inputs from vector *A*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vfill

Vector fill; single precision.

```
void
vDSP_vfill (float * A,
float * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Single-precision real input scalar

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_{nK} = A \quad n = \{0, N-1\}$$

Sets each element of vector *C* to the value of *A*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vfillD

Vector fill; double precision.

```
void
vDSP_vfillD (double * A,
             double * C,
             vDSP_Stride K,
             vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input scalar
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

Performs the operation

$$C_{nK} = A \quad n = \{0, N-1\}$$

Sets each element of vector *C* to the value of *A*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vfilli

Integer vector fill.

```
void
vDSP_vfilli (int * A,
             int * C,
             vDSP_Stride K,
             vDSP_Length N);
```

Parameters

A
Integer input scalar

C
Integer output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_{nK} = A \quad n = \{0, N-1\}$$

Sets each element of vector *C* to the value of *A*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vfrac

Vector truncate to fraction; single precision.

```
void vDSP_vfrac (float * A,
                 vDSP_Stride I,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} - \text{truncate}(A_{nI}) \quad n = \{0, N-1\}$$

The "function" `truncate(x)` is the integer farthest from 0 but not farther than x. Thus, for example, `vDSP_vFrac(-3.25)` produces the result `-0.25`.

Sets each element of vector C to the signed fractional part of the corresponding element of A.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_vfracD

Vector truncate to fraction; double precision.

```
void vDSP_vfracD (double * A,
                 vDSP_Stride I,
                 double * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} - \text{truncate}(A_{nI}) \quad n = \{0, N-1\}$$

The "function" `truncate(x)` is the integer farthest from 0 but not farther than x. Thus, for example, `vDSP_vFrac(-3.25)` produces the result `-0.25`.

Sets each element of vector C to the signed fractional part of the corresponding element of A.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgather

Vector gather; single precision.

```
void vDSP_vgather (float * A,
                  vDSP_Length * B,
                  vDSP_Stride J,
                  float * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>B</i>	Integer vector containing indices
<i>J</i>	Stride for <i>B</i>
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

Performs the operation

$$C_{nK} = A_{B_{nJ}} \quad n = \{0, N-1\}$$

Uses elements of vector *B* as indices to copy selected elements of vector *A* to sequential locations in vector *C*. Note that 1, not zero, is treated as the first location in the input vector when evaluating indices. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgathera

Vector gather, absolute pointers; single precision.

```
void vDSP_vgather (float ** A,
                  vDSP_Stride I,
                  float * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

A
Pointer input vector

I
Stride for *A*

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_{nK} = *(A_{nI}) \quad n = \{0, N-1\}$$

Uses elements of vector *A* as pointers to copy selected single-precision values from memory to sequential locations in vector *C*. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgatherD

Vector gather, absolute pointers; double precision.

```
void vDSP_vgatherD (double ** A,
                   vDSP_Stride I,
                   double * C,
                   vDSP_Stride K,
                   vDSP_Length N);
```

Parameters

A
Pointer input vector

I
Stride for *A*

C
Double-precision real output vector

K
Stride for C

N
Count

Discussion

Performs the operation

$$C_{nK} = *(A_{nI}) \quad n = \{0, N-1\}$$

Uses elements of vector *A* as pointers to copy selected double-precision values from memory to sequential locations in vector *C*. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgatherD

Vector gather; double precision.

```
void vDSP_vgatherD (double * A,
vDSP_Length * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Double-precision real input vector

B
Integer vector containing indices

J
Stride for B

C
Double-precision real output vector

K
Stride for C

N
Count

Discussion

Performs the operation

$$C_{nK} = A_{B_{nI}} \quad n = \{0, N-1\}$$

Uses elements of vector *B* as indices to copy selected elements of vector *A* to sequential locations in vector *C*. Note that 1, not zero, is treated as the first location in the input vector when evaluating indices. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgen

Vector tapered ramp; single precision.

```
void vDSP_vgen (float * A,
               float * B,
               float * C,
               vDSP_Stride K,
               vDSP_Length N);
```

Parameters

A
Single-precision real input scalar: base value

B
Single-precision real input scalar: end value

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_{nK} = A + \frac{n(B-A)}{N-1} \quad n = \{0, N-1\}$$

Creates ramped vector *C* with element zero equal to scalar *A* and element *N*-1 equal to scalar *B*. Output values between element zero and element *N*-1 are evenly spaced and increase or decrease monotonically.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgenD

Vector tapered ramp; double precision.

```
void vDSP_vgenD (double * A,
double * B,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Double-precision real input scalar: base value

B
Double-precision real input scalar: end value

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_{nK} = A + \frac{n(B-A)}{N-1} \quad n = \{0, N-1\}$$

Creates ramped vector *C* with element zero equal to scalar *A* and element *N*-1 equal to scalar *B*. Output values between element zero and element *N*-1 are evenly spaced and increase or decrease monotonically.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgenp

Vector generate by extrapolation and interpolation; single precision.

```
void vDSP_vgenp (float * A,
vDSP_Stride I,
float * B,
vDSP_Stride J,
float * C,
vDSP_Stride K,
vDSP_Length N,
vDSP_Length M);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

B	Single-precision real input vector
J	Stride for B
C	Single-precision real output vector
K	Stride for C
N	Count for C
M	Count for A and B

Discussion

Performs the operation

$$C_{nK} = A_0 \quad \text{for } 0 \leq n \leq \text{trunc}(B_0)$$

$$C_{nK} = A_{[M-1]I} \quad \text{for } \text{trunc}(B_{[M-1]J}) < n \leq N-1$$

$$C_{nK} = A_{mI} + \frac{A_{\Delta} (n - B_{mJ})}{B_{\Delta}} \quad \text{for } \text{trunc}(B_{mJ}) < n \leq \text{trunc}(B_{[m+1]J})$$

where: $A_{\Delta} = A_{[m+1]I} - A_{mI}$

$$B_{\Delta} = B_{[m+1]J} - B_{mJ} \quad m = \{0, M-2\}$$

Generates vector C by extrapolation and linear interpolation from the ordered pairs (A,B) provided by corresponding elements in vectors A and B. Vector B provides index values and should increase monotonically. Vector A provides intensities, magnitudes, or some other measurable quantities, one value associated with each value of B. This function can only be done out of place.

Vectors A and B define a piecewise linear function, $f(x)$:

- In the interval $[-\infty, \text{trunc}(B[0*J])]$, the function is the constant $A[0*I]$.
- In each interval $(\text{trunc}(B[m*J]), \text{trunc}(B[(m+1)*J]))$, the function is the line passing through the two points $(B[m*J], A[m*I])$ and $(B[(m+1)*J], A[(m+1)*I])$. (This is for each integer m , $0 \leq m < M-1$.)
- In the interval $(B[(M-1)*J], \infty)$, the function is the constant $A[(M-1)*I]$.
- For $0 \leq n < N$, $C[n*K] = f(n)$.

This function can only be done out of place.

Output values are generated for integral indices in the range zero through $N - 1$, deriving output values by interpolating and extrapolating from vectors A and B. For example, if vectors A and B define velocity and time pairs (v, t) , `vDSP_vgenp` writes one velocity to vector C for every integral unit of time from zero to $N - 1$.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vgenpD

Vector generate by extrapolation and interpolation; double precision.

```
void vDSP_vgenpD (double * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
vDSP_Length N,
vDSP_Length M);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input vector
<i>J</i>	Stride for B
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count for C
<i>M</i>	Count for A and B

Discussion

Performs the operation

$$C_{nK} = A_0 \quad \text{for } 0 \leq n \leq \text{trunc}(B_0)$$

$$C_{nK} = A_{[M-1]I} \quad \text{for } \text{trunc}(B_{[M-1]J}) < n \leq N-1$$

$$C_{nK} = A_{mI} + \frac{A_{\Delta} (n - B_{mJ})}{B_{\Delta}} \quad \text{for } \text{trunc}(B_{mJ}) < n \leq \text{trunc}(B_{[m+1]J})$$

where: $A_{\Delta} = A_{[m+1]I} - A_{mI}$

$$B_{\Delta} = B_{[m+1]J} - B_{mJ} \quad m = \{0, M-2\}$$

Generates vector *C* by extrapolation and linear interpolation from the ordered pairs (*A*,*B*) provided by corresponding elements in vectors *A* and *B*. Vector *B* provides index values and should increase monotonically. Vector *A* provides intensities, magnitudes, or some other measurable quantities, one value associated with each value of *B*. This function can only be done out of place.

Vectors *A* and *B* define a piecewise linear function, *f*(*x*):

- In the interval $[-\infty, \text{trunc}(B[0*J])]$, the function is the constant $A[0*I]$.
- In each interval $(\text{trunc}(B[m*J]), \text{trunc}(B[(m+1)*J]))$, the function is the line passing through the two points $(B[m*J], A[m*I])$ and $(B[(m+1)*J], A[(m+1)*I])$. (This is for each integer *m*, $0 \leq m < M-1$.)
- In the interval $(B[(M-1)*J], \infty)$, the function is the constant $A[(M-1)*I]$.
- For $0 \leq n < N$, $C[n*K] = f(n)$.

This function can only be done out of place.

Output values are generated for integral indices in the range zero through *N* - 1, deriving output values by interpolating and extrapolating from vectors *A* and *B*. For example, if vectors *A* and *B* define velocity and time pairs (*v*, *t*), `vDSP_vgenp` writes one velocity to vector *C* for every integral unit of time from zero to *N* - 1.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_viclip

Vector inverted clip; single precision.

```
void vDSP_viclip (float * A,
                 vDSP_Stride I,
                 float * B,
                 float * C,
                 float * D,
                 vDSP_Stride L,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for <i>A</i>
<i>B</i>	Single-precision real input scalar: lower threshold
<i>C</i>	Single-precision real input scalar: upper threshold
<i>D</i>	Single-precision real output vector

L
Stride for D

N
Count

Discussion

Performs the operation

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \leq b$$

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \geq c$$

$$D_{nj} = b \quad \text{if} \quad b < A_{ni} < 0.0$$

$$D_{nj} = c \quad \text{if} \quad 0.0 \leq A_{ni} < c \quad n = \{0, N-1\}$$

Performs an inverted clip of vector A using lower-threshold and upper-threshold input scalars B and C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_viclipD

Vector inverted clip; double precision.

```
void vDSP_viclipD (double * A,
vDSP_Stride I,
double * B,
double * C,
double * D,
vDSP_Stride L,
vDSP_Length N);
```

Parameters

A
Double-precision real input vector

I
Stride for A

B
Double-precision real input scalar: lower threshold

C
Double-precision real input scalar: upper threshold

D
Double-precision real output vector

L
Stride for D

N

Count

Discussion

Performs the operation

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \leq b$$

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \geq c$$

$$D_{nj} = b \quad \text{if} \quad b < A_{ni} < 0.0$$

$$D_{nj} = c \quad \text{if} \quad 0.0 \leq A_{ni} < c \quad n = \{0, N-1\}$$

Performs an inverted clip of vector *A* using lower-threshold and upper-threshold input scalars *B* and *C*.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vindex

Vector index; single precision.

```
void vDSP_vindex (float * A,
                 float * B,
                 vDSP_Stride J,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters*A*

Single-precision real input vector

B

Single-precision real input vector: indices

*J*Stride for *B**C*

Single-precision real output vector

*K*Stride for *C**N*

Count

Discussion

Performs the operation

$$C_{nK} = A_{\text{truncate}(B_{nJ})} \quad n = \{0, N-1\}$$

Uses vector *B* as zero-based subscripts to copy selected elements of vector *A* to vector *C*. Fractional parts of vector *B* are ignored.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vindexD

Vector index; double precision.

```
void vDSP_vindexD (double * A,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>B</i>	Double-precision real input vector: indices
<i>J</i>	Stride for <i>B</i>
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

Performs the operation

$$C_{nK} = A_{\text{truncate}(B_{nJ})} \quad n = \{0, N-1\}$$

Uses vector *B* as zero-based subscripts to copy selected elements of vector *A* to vector *C*. Fractional parts of vector *B* are ignored.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vlim

Vector test limit; single precision.

```
void vDSP_vlim (float * A,
               vDSP_Stride I,
               float * B,
               float * C,
               float * D,
               vDSP_Stride L,
               vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Single-precision real input scalar: limit
<i>C</i>	Single-precision real input scalar
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Compares values from vector *A* to limit scalar *B*. For inputs greater than or equal to *B*, scalar *C* is written to *D*. For inputs less than *B*, the negated value of scalar *C* is written to vector *D*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vlimD

Vector test limit; double precision.

```
void vDSP_vlimD (double * A,
vDSP_Stride I,
double * B,
double * C,
double * D,
vDSP_Stride L,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input scalar: limit
<i>C</i>	Double-precision real input scalar
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Compares values from vector *A* to limit scalar *B*. For inputs greater than or equal to *B*, scalar *C* is written to *D*. For inputs less than *B*, the negated value of scalar *C* is written to vector *D*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vlint

Vector linear interpolation between neighboring elements; single precision.

```
void vDSP_vlint (float * A,
float * B,
vDSP_Stride J,
float * C,
vDSP_Stride K,
vDSP_Length N,
vDSP_Length M);
```

Parameters

<i>A</i>	Single-precision real input vector
----------	------------------------------------

<i>B</i>	Single-precision real input vector: integer parts are indices into A and fractional parts are interpolation constants
<i>J</i>	Stride for B
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count for C
<i>M</i>	Length of A

Discussion

Performs the operation

$$C_{nK} = A_{\beta} + \alpha(A_{\beta+1} - A_{\beta}) \quad n = \{0, N-1\}$$

where: $\beta = \text{trunc}(B_{nJ})$

$$\alpha = B_{nJ} - \text{float}(\beta)$$

Generates vector C by interpolating between neighboring values of vector A as controlled by vector B. The integer portion of each element in B is the zero-based index of the first element of a pair of adjacent values in vector A.

The value of the corresponding element of C is derived from these two values by linear interpolation, using the fractional part of the value in B.

Argument M is not used in the calculation. However, the integer parts of the values in B must be greater than or equal to zero and less than or equal to M - 2.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vlintD

Vector linear interpolation between neighboring elements; double precision.

```
void vDSP_vlintD (double * A,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
vDSP_Length N,
vDSP_Length M);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>B</i>	Double-precision real input vector: integer parts are indices into <i>A</i> and fractional parts are interpolation constants
<i>J</i>	Stride for <i>B</i>
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count for <i>C</i>
<i>M</i>	Length of <i>A</i>

Discussion

Performs the operation

$$C_{nK} = A_{\beta} + \alpha(A_{\beta+1} - A_{\beta}) \quad n = \{0, N-1\}$$

where: $\beta = \text{trunc}(B_{nJ})$

$$\alpha = B_{nJ} - \text{float}(\beta)$$

Generates vector *C* by interpolating between neighboring values of vector *A* as controlled by vector *B*. The integer portion of each element in *B* is the zero-based index of the first element of a pair of adjacent values in vector *A*.

The value of the corresponding element of *C* is derived from these two values by linear interpolation, using the fractional part of the value in *B*.

Argument *M* is not used in the calculation. However, the integer parts of the values in *B* must be greater than or equal to zero and less than or equal to *M* - 2.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vnabs

Vector negative absolute values; single precision.

```
void vDSP_vnabs (float * A,
                vDSP_Stride I,
                float * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

This performs the operation

$$C_{nK} = -|A_{nI}| \quad n = \{0, N-1\}$$

Each value in C is the negated absolute value of the corresponding element in A.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vnabsD

Vector negative absolute values; double precision.

```
void vDSP_vnabsD (double * A,
                 vDSP_Stride I,
                 double * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

This performs the operation

$$C_{nK} = -|A_{nI}| \quad n = \{0, N-1\}$$

Each value in *C* is the negated absolute value of the corresponding element in *A*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vneg

Vector negative values; single precision.

```
void
vDSP_vneg (float * A,
vDSP_Stride I,
float * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

Each value in *C* is the negated value of the corresponding element in *A*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vnegD

Vector negative values; double precision.

```
void
vDSP_vnegD (double * A,
vDSP_Stride I,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Each value in C is the negated value of the corresponding element in A.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vramp

Build ramped vector; single precision.

```
void
vDSP_vramp (float * A,
float * B,
float * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input scalar: initial value
<i>B</i>	Single-precision real input scalar: increment or decrement
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C

N

Count

Discussion

Performs the operation

$$C_{nk} = a + nb \quad n = \{0, N-1\}$$

Creates a monotonically incrementing or decrementing vector. Scalar A is the initial value written to vector C. Scalar B is the increment or decrement for each succeeding element.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vrampD

Build ramped vector; double precision.

```
void
vDSP_vrampD (double * A,
             double * B,
             double * C,
             vDSP_Stride K,
             vDSP_Length N);
```

Parameters*A*

Double-precision real input scalar: initial value

B

Double-precision real input scalar: increment or decrement

C

Double-precision real output vector

K

Stride for C

N

Count

Discussion

Performs the operation

$$C_{nk} = a + nb \quad n = \{0, N-1\}$$

Creates a monotonically incrementing or decrementing vector. Scalar A is the initial value written to vector C. Scalar B is the increment or decrement for each succeeding element.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vrsum

Vector running sum integration; single precision.

```
void vDSP_vrsum (float * A,
                vDSP_Stride I,
                float * S,
                float * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters*A*

Single-precision real input vector

*I*Stride for *A**S*

Single-precision real input scalar: weighting factor

C

Single-precision real output vector

*K*Stride for *C**N*

Count

Discussion

Performs the operation

$$C_0 = 0$$

$$C_{mK} = C_{(m-1)K} + SA_{mI} \quad m = \{1, N-1\}$$

Integrates vector *A* using a running sum from vector *C*. Vector *A* is weighted by scalar *S* and added to the previous output point. The first element from vector *A* is not used in the sum.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vrsumD

Vector running sum integration; double precision.

```
void vDSP_vrsumD (double * A,
                 vDSP_Stride I,
                 double * S,
                 double * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>S</i>	Double-precision real input scalar: weighting factor
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

$$C_0 = 0$$

$$C_{mK} = C_{(m-1)K} + SA_{mI} \quad m = \{1, N-1\}$$

Integrates vector A using a running sum from vector C. Vector A is weighted by scalar S and added to the previous output point. The first element from vector A is not used in the sum.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vrvrs

Vector reverse order, in place; single precision.

```
void vDSP_vrvrs (float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>C</i>	Single-precision real input-output vector
<i>K</i>	Stride for C

N
Count

Discussion

Performs the operation

$$C_{nK} \leftrightarrow C_{[N-n-1]K} \quad n = \{0, (N/2)-1\}$$

Reverses the order of vector C in place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vrvrsD

Vector reverse order, in place; double precision.

```
void vDSP_vrvrsD (double * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

C
Double-precision real input-output vector

K
Stride for C

N
Count

Discussion

Performs the operation

$$C_{nK} \leftrightarrow C_{[N-n-1]K} \quad n = \{0, (N/2)-1\}$$

Reverses the order of vector C in place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsimps

Simpson integration; single precision.

```
void vDSP_vsimps (float * A,
                 vDSP_Stride I,
                 float * B,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Single-precision real input scalar
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_K = \frac{[A_0 + A_I]B}{2}$$

$$C_{nK} = C_{[n-2]K} + \frac{[A_{[n-2]I} + 4.0 \times A_{[n-1]I} + A_{nI}]B}{3} \quad n = \{2, N-1\}$$

Integrates vector A using Simpson integration, storing results in vector C. Scalar B specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsimpsD

Simpson integration; double precision.

```
void vDSP_vsimpsD (double * A,
vDSP_Stride I,
double * B,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input scalar
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_K = \frac{[A_0 + A_I]B}{2}$$

$$C_{nK} = C_{[n-2]K} + \frac{[A_{[n-2]I} + 4.0 \times A_{[n-1]I} + A_{nI}]B}{3} \quad n = \{2, N-1\}$$

Integrates vector A using Simpson integration, storing results in vector C. Scalar B specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsort

Vector in-place sort; single precision.

```
void vDSP_vsort (float * C,
vDSP_Length N,
int OFLAG);
```

Parameters

<i>C</i>	Single-precision real input-output vector
----------	---

N

Count

OFLAG

Flag for sort order: 1 for ascending, -1 for descending

DiscussionPerforms an in-place sort of vector *C* in the order specified by parameter *OFLAG*.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsortD

Vector in-place sort; double precision.

```
void vDSP_vsortD (double * C,
                 vDSP_Length N,
                 int OFLAG);
```

Parameters*C*

Double-precision real input-output vector

N

Count

OFLAG

Flag for sort order: 1 for ascending, -1 for descending

DiscussionPerforms an in-place sort of vector *C* in the order specified by parameter *OFLAG*.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsorti

Vector index in-place sort; single precision.

```
void vDSP_vsorti (float * C,
                 vDSP_Length * IC,
                 vDSP_Length * List_addr,
                 vDSP_Length N,
                 int OFLAG);
```

Parameters*C*

Single-precision real input vector

*IC*Integer output vector. Must be initialized with the indices of vector *C*, from 0 to *N*-1.*List_addr*

Temporary vector. This is currently not used and NULL should be passed.

N

Count

OFLAG

Flag for sort order: 1 for ascending, -1 for descending

DiscussionLeaves input vector *C* unchanged and performs an in-place sort of the indices in vector *IC* according to the values in *C*. The sort order is specified by parameter *OFLAG*.The values in *C* can then be obtained in sorted order, by taking indices in sequence from *IC*.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsortiD

Vector index in-place sort; double precision.

```
void vDSP_vsortiD (double * C,
                  vDSP_Length * IC,
                  vDSP_Length * List_addr,
                  vDSP_Length N,
                  int OFLAG);
```

Parameters*C*

Double-precision real input vector

*IC*Integer output vector. Must be initialized with the indices of vector *C*, from 0 to *N*-1.*List_addr*

Temporary vector. This is currently not used and NULL should be passed.

N

Count

OFLAG

Flag for sort order: 1 for ascending, -1 for descending

DiscussionLeaves input vector *C* unchanged and performs an in-place sort of the indices in vector *IC* according to the values in *C*. The sort order is specified by parameter *OFLAG*.The values in *C* can then be obtained in sorted order, by taking indices in sequence from *IC*.**Availability**

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vspdp

Vector convert single-precision to double-precision.

```
void vDSP_vspdp (float * A,
                vDSP_Stride I,
                double * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

This performs the operation

$$C_{nk} = A_{ni} \quad n = \{0, N-1\}$$

Creates double-precision vector *C* by converting single-precision inputs from vector *A*. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsqComputes the squared values of vector *input* and leaves the result in vector *result*; single precision.

```
void vDSP_vsq (const float input[],
              vDSP_Stride strideInput,
              float result[],
              vDSP_Stride strideResult,
              vDSP_Length size);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI}^2 \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsqD

Computes the squared values of vector `signal1` and leaves the result in vector `result`; double precision.

```
void vDSP_vsqD (const double input[],
               vDSP_Stride strideInput,
               double result[],
               vDSP_Stride strideResult,
               vDSP_Length size);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI}^2 \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vssq

Computes the signed squares of vector `signal1` and leaves the result in vector `result`; single precision.

```
void vDSP_vssq (const float input[],
               vDSP_Stride strideInput,
               float result[],
               vDSP_Stride strideResult,
               vDSP_Length size);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} \cdot |A_{nI}| \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vssqD

Computes the signed squares of vector `signal` and leaves the result in vector `result`; double precision.

```
void vDSP_vssqD (const double input[],
                vDSP_Stride strideInput,
                double result[],
                vDSP_Stride strideResult,
                vDSP_Length size);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} \cdot |A_{nI}| \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vswsum

Vector sliding window sum; single precision.

```
void vDSP_vswsum (float * A,
                 vDSP_Stride I,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N,
                 vDSP_Length P);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count of output points
<i>P</i>	Length of window

Discussion

Performs the operation

$$C_0(P) = \sum_{p=0}^{P-1} A_{qi} (C_{nk}(P) - C_{(n-1)k}(P) + A_{(n+P-1)i} - A_{(n-1)i}) \quad n = \{1, N-1\}$$

Writes the sliding window sum of P consecutive elements of vector A to vector C , for each of N possible starting positions of the P -element window in vector A .

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vswsumD

Vector sliding window sum; double precision.

```
void vDSP_vswsumD (double * A,
                  vDSP_Stride I,
                  double * C,
                  vDSP_Stride K,
                  vDSP_Length N,
                  vDSP_Length P);
```

Parameters

A	Double-precision real input vector
I	Stride for A
C	Double-precision real output vector
K	Stride for C
N	Count of output points
P	Length of window

Discussion

Performs the operation

$$C_0(P) = \sum_{p=0}^{P-1} A_{qi} \quad (C_{nk}(P) - C_{(n-1)k}(P) + A_{(n+P-1)i} - A_{(n-1)i}) \quad n = \{1, N-1\}$$

Writes the sliding window sum of P consecutive elements of vector A to vector C , for each of N possible starting positions of the P -element window in vector A .

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vtabi

Vector interpolation, table lookup; single precision.

```
void vDSP_vtabi (float * A,
                vDSP_Stride I,
                float * S1,
                float * S2,
                float * C,
                vDSP_Length M,
                float * D,
                vDSP_Stride L,
                vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for <i>A</i>
<i>S1</i>	Single-precision real input scalar: scale factor
<i>S2</i>	Single-precision real input scalar: base offset
<i>C</i>	Single-precision real input vector: lookup table
<i>M</i>	Lookup table size
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for <i>D</i>
<i>N</i>	Count

Discussion

Performs the operation

$$p = F \cdot A_{ni} + G \quad q = \text{floor}(p) \quad r = p - \text{float}(q) \quad D_{nk} = (1.0-r)C_q + rC_{q-1} \quad n = \{0, N-1\}$$

Evaluates elements of vector *A* for use as offsets into vector *C*. Vector *C* is a zero-based lookup table supplied by the caller that generates output values for vector *D*. Linear interpolation is used to compute output values when offsets do not evaluate integrally. Scale factor *S1* and base offset *S2* map the anticipated range of input values to the range of the lookup table and are typically assigned values such that:

$$\begin{aligned} \text{floor}(F \cdot \text{minimum input value} + G) &= 0 \\ \text{floor}(F \cdot \text{maximum input value} + G) &= M-1 \end{aligned}$$

Input values that evaluate to zero or less derive their output values from table location zero. Values that evaluate beyond the table, greater than M-1, derive their output values from the last table location. For inputs that evaluate integrally, the table location indexed by the integral is copied as the output value. All other inputs derive their output values by interpolation between the two table values surrounding the evaluated input.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vtabiD

Vector interpolation, table lookup; double precision.

```
void vDSP_vtabiD (double * A,
                 vDSP_Stride I,
                 double * S1,
                 double * S2,
                 double * C,
                 vDSP_Length M,
                 double * D,
                 vDSP_Stride L,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>S1</i>	Double-precision real input scalar: scale factor
<i>S2</i>	Double-precision real input scalar: base offset
<i>C</i>	Double-precision real input vector: lookup table
<i>M</i>	Lookup table size
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

$$p = F \cdot A_{nl} + G \quad q = \text{floor}(p) \quad r = p - \text{float}(q) \quad D_{nk} = (1.0-r)C_q + rC_{q-1} \quad n = \{0, N-1\}$$

Evaluates elements of vector *A* for use as offsets into vector *C*. Vector *C* is a zero-based lookup table supplied by the caller that generates output values for vector *D*. Linear interpolation is used to compute output values when offsets do not evaluate integrally. Scale factor *S1* and base offset *S2* map the anticipated range of input values to the range of the lookup table and are typically assigned values such that:

```
floor(F * minimum input value + G) = 0
floor(F * maximum input value + G) = M-1
```

Input values that evaluate to zero or less derive their output values from table location zero. Values that evaluate beyond the table, greater than *M-1*, derive their output values from the last table location. For inputs that evaluate integrally, the table location indexed by the integral is copied as the output value. All other inputs derive their output values by interpolation between the two table values surrounding the evaluated input.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vthr

Vector threshold; single precision.

```
void vDSP_vthr (float * A,
               vDSP_Stride I,
               float * B,
               float * C,
               vDSP_Stride K,
               vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for <i>A</i>
<i>B</i>	Single-precision real input scalar: lower threshold
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $C_{nK} = A_{nI}$ else $C_{nK} = B$ $n = \{0, N-1\}$

Creates vector *C* by comparing each input from vector *A* with scalar *B*. If an input value is less than *B*, *B* is copied to *C*; otherwise, the input value from *A* is copied to *C*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vthrD

Vector threshold; double precision.

```
void vDSP_vthrD (double * A,
                vDSP_Stride I,
                double * B,
                double * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for <i>A</i>
<i>B</i>	Double-precision real input scalar: lower threshold
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

Performs the operation

$$\text{If } A_{nI} \geq B \text{ then } C_{nK} = A_{nI} \text{ else } C_{nK} = B \quad n = \{0, N-1\}$$

Creates vector *C* by comparing each input from vector *A* with scalar *B*. If an input value is less than *B*, *B* is copied to *C*; otherwise, the input value from *A* is copied to *C*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vthres

Vector threshold with zero fill; single precision.

```
void vDSP_vthres (float * A,
                 vDSP_Stride I,
                 float * B,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Single-precision real input scalar: lower threshold
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $C_{nK} = A_{nI}$ else $C_{nK} = 0.0$ $n = \{0, N-1\}$

Creates vector C by comparing each input from vector A with scalar B. If an input value is less than B, zero is written to C; otherwise, the input value from A is copied to C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vthresD

Vector threshold with zero fill; double precision.

```
void vDSP_vthresD (double * A,
vDSP_Stride I,
double * B,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input scalar: lower threshold
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $C_{nK} = A_{nI}$ else $C_{nK} = 0.0$ $n = \{0, N-1\}$

Creates vector C by comparing each input from vector A with scalar B. If an input value is less than B, zero is written to C; otherwise, the input value from A is copied to C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vthrsc

Vector threshold with signed constant; single precision.

```
void vDSP_vthrsc (float * A,
vDSP_Stride I,
float * B,
float * C,
float * D,
vDSP_Stride L,
vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector
----------	------------------------------------

<i>I</i>	Stride for A
<i>B</i>	Single-precision real input scalar: lower threshold
<i>C</i>	Single-precision real input scalar
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $D_{nM} = C$ else $D_{nM} = -C$ $n = \{0, N-1\}$

Creates vector D using the plus or minus value of scalar C. The sign of the output element is determined by comparing input from vector A with threshold scalar B. For input values less than B, the negated value of C is written to vector D. For input values greater than or equal to B, C is copied to vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vthrscD

Vector threshold with signed constant; double precision.

```
void vDSP_vthrscD (double * A,
vDSP_Stride I,
double * B,
double * C,
double * D,
vDSP_Stride L,
vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input scalar: lower threshold
<i>C</i>	Double-precision real input scalar

D
Double-precision real output vector

L
Stride for *D*

N
Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $D_{nM} = C$ else $D_{nM} = -C$ $n = \{0, N-1\}$

Creates vector *D* using the plus or minus value of scalar *C*. The sign of the output element is determined by comparing input from vector *A* with threshold scalar *B*. For input values less than *B*, the negotiated value of *C* is written to vector *D*. For input values greater than or equal to *B*, *C* is copied to vector *D*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vtrapz

Vector trapezoidal integration; single precision.

```
void vDSP_vtrapz (float * A,
vDSP_Stride I,
float * B,
float * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

B
Single-precision real input scalar: step size

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_{nK} = C_{[n-1]K} + \frac{B[A_{[n-1]I} + A_{nI}]}{2} \quad n = \{1, N-1\}$$

Estimates the integral of vector A using the trapezoidal rule. Scalar B specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vtrapzD

Vector trapezoidal integration; double precision.

```
void vDSP_vtrapzD (double * A,
                  vDSP_Stride I,
                  double * B,
                  double * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input scalar: step size
<i>C</i>	Double-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_{nK} = C_{[n-1]K} + \frac{B[A_{[n-1]I} + A_{nI}]}{2} \quad n = \{1, N-1\}$$

Estimates the integral of vector A using the trapezoidal rule. Scalar B specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvabs

Complex vector absolute values; single precision.

```
void vDSP_zvabs (DSPSplitComplex * A,
                vDSP_Stride I,
                float * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision complex input vector
<i>I</i>	Stride for <i>A</i>
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

This performs the operation

$$C_{nk} = \sqrt{\operatorname{Re}[A_{ni}]^2 + \operatorname{Im}[A_{ni}]^2} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvabsD

Complex vector absolute values; double precision.

```
void vDSP_zvabsD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Double-precision complex input vector

I
Stride for *A*

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

This performs the operation

$$C_{nk} = \sqrt{\operatorname{Re}[A_{ni}]^2 + \operatorname{Im}[A_{ni}]^2} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvconj

Complex vector conjugate; single precision.

```
void vDSP_zvconj (DSPSplitComplex * A,
vDSP_Stride I,
DSPSplitComplex * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Single-precision complex input vector

I
Stride for *A*

C
Single-precision complex output vector

K
Stride for *C*

N
Count

Discussion

Conjugates vector A.

$$C_{nk} = A_{ni}^*$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvconjD

Complex vector conjugate; double precision.

```
void vDSP_zvconjD (DSPDoubleSplitComplex * A,
                  vDSP_Stride I,
                  DSPDoubleSplitComplex * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

A
Double-precision complex input vector

I
Stride for A

C
Double-precision complex output vector

K
Stride for C

N
Count

Discussion

Conjugates vector A.

$$C_{nk} = A_{ni}^*$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvfill

Complex vector fill; single precision.

```
void
vDSP_zvfill (DSPSplitComplex * A,
             DSPSplitComplex * C,
             vDSP_Stride K,
             vDSP_Length N);
```

Parameters

A
Single-precision complex input scalar

C
Single-precision complex output vector

K
Stride for *C*

N
Count

Discussion

Sets each element in complex vector *C* to complex scalar *A*.

$$C_{nK} = A \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvfillD

Complex vector fill; double precision.

```
void
vDSP_zvfillD (DSPDoubleSplitComplex * A,
              DSPDoubleSplitComplex * C,
              vDSP_Stride K,
              vDSP_Length N);
```

Parameters

A
Double-precision complex input scalar

C
Double-precision complex output vector

K
Stride for *C*

N
Count

Discussion

Sets each element in complex vector *C* to complex scalar *A*.

$$C_{nK} = A \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvmags

Complex vector magnitudes squared; single precision.

```
void vDSP_zvmags (DSPSplitComplex * A,
                 vDSP_Stride I,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision complex input vector
<i>I</i>	Stride for <i>A</i>
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

Calculates the squared magnitudes of complex vector *A*.

$$C_{nK} = (Re[A_{nI}])^2 + (Im[A_{nI}])^2 \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvmagsD

Complex vector magnitudes squared; double precision.

```
void vDSP_zvmagsD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Double-precision complex input vector

I
Stride for *A*

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

Calculates the squared magnitudes of complex vector *A*.

$$C_{nK} = (Re[A_{nI}])^2 + (Im[A_{nI}])^2 \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvmgsa

Complex vector magnitudes square and add; single precision.

```
void vDSP_zvmgsa (DSPSplitComplex * A,
vDSP_Stride I,
float * B,
vDSP_Stride J,
float * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Single-precision complex input vector

I
Stride for *A*

B
Single-precision real input vector

J
Stride for *B*

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion

Adds the squared magnitudes of complex vector *A* to real vector *B* and store the results in real vector *C*.

$$C_{nK} = [Re[A_{nI}]]^2 + [Im[A_{nI}]]^2 + B_{nJ} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvmgsaD

Complex vector magnitudes square and add; double precision.

```
void vDSP_zvmgsaD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Double-precision complex input vector

I
Stride for *A*

B
Double-precision real input vector

J
Stride for *B*

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

Adds the squared magnitudes of complex vector *A* to real vector *B* and store the results in real vector *C*.

$$C_{nK} = [Re[A_{nI}]]^2 + [Im[A_{nI}]]^2 + B_{nJ} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zmov

Complex vector copy; single precision.

```
void vDSP_zmov (DSPSplitComplex * A,
               vDSP_Stride I,
               DSPSplitComplex * C,
               vDSP_Stride K,
               vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision complex input vector
<i>I</i>	Stride for A
<i>C</i>	Single-precision complex output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Copies complex vector A to complex vector C.

$$C_{nK} = A_{nI}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zmovD

Complex vector copy; double precision.

```
void vDSP_zvmovD (DSPDoubleSplitComplex * A,
                 vDSP_Stride I,
                 DSPDoubleSplitComplex * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision complex input vector
<i>I</i>	Stride for A
<i>C</i>	Double-precision complex output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Copies complex vector A to complex vector C.

$$C_{nK} = A_{nI}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvneg

Complex vector negate; single precision.

```
void vDSP_zvneg (DSPSplitComplex * A,
                vDSP_Stride I,
                DSPSplitComplex * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision complex input vector
<i>I</i>	Stride for A
<i>C</i>	Single-precision complex output vector
<i>K</i>	Stride for C

N
Count

Discussion

Computes the negatives of the values of complex vector *A* and puts them into complex vector *C*.

$$C_{nK} = -A_{nI}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvnegD

Complex vector negate; double precision.

```
void
vDSP_zvnegD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
DSPDoubleSplitComplex * C,
vDSP_Stride K,
vDSP_Length N);
```

Parameters

A
Double-precision complex input vector

I
Stride for *A*

C
Double-precision complex output vector

K
Stride for *C*

N
Count

Discussion

Computes the negatives of the values of complex vector *A* and puts them into complex vector *C*.

$$C_{nK} = -A_{nI}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvphas

Complex vector phase; single precision.

```
void vDSP_zvphas (DSPSplitComplex * A,
                 vDSP_Stride I,
                 float * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision complex input vector
<i>I</i>	Stride for <i>A</i>
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for <i>C</i>
<i>N</i>	Count

Discussion

Finds the phase values, in radians, of complex vector *A* and store the results in real vector *C*. The results are between $-\pi$ and $+\pi$. The sign of the result is the sign of the second coordinate in the input vector.

$$C_{nK} = \operatorname{atan} \frac{\operatorname{Im}[A_n]}{\operatorname{Re}[A_n]} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvphasD

Complex vector phase; double precision.

```
void vDSP_zvphasD (DSPDoubleSplitComplex * A,
                  vDSP_Stride I,
                  double * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision complex input vector
<i>I</i>	Stride for <i>A</i>

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

Finds the phase values, in radians, of complex vector *A* and store the results in real vector *C*. The results are between $-\pi$ and $+\pi$. The sign of the result is the sign of the second coordinate in the input vector.

$$C_{nK} = \operatorname{atan} \frac{\operatorname{Im}[A_{nI}]}{\operatorname{Re}[A_{nI}]} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

Document Revision History

This table describes the changes to *vDSP Single-Vector Operations Reference*.

Date	Notes
2009-01-07	TBD
2009-01-06	TBD
2008-11-19	Corrected misplacement of matrix transposition function.
2007-06-15	New document that describes the C API for the vDSP functions that operate on the elements of a single vector

REVISION HISTORY

Document Revision History

Index

V

- vDSP_nzcros function 13
- vDSP_nzcrosD function 14
- vDSP_polar function 16
- vDSP_polarD function 16
- vDSP_rect function 17
- vDSP_rectD function 18
- vDSP_vabs function 19
- vDSP_vabsD function 20
- vDSP_vabsi function 20
- vDSP_vavlin function 21
- vDSP_vavlinD function 22
- vDSP_vclip function 23
- vDSP_vclipc function 24
- vDSP_vclipcD function 25
- vDSP_vclipD function 26
- vDSP_vclr function 27
- vDSP_vclrD function 27
- vDSP_vcmprs function 27
- vDSP_vcmprsD function 28
- vDSP_vdbcon function 29
- vDSP_vdbconD function 30
- vDSP_vdpsp function 31
- vDSP_vfill function 32
- vDSP_vfillD function 33
- vDSP_vfilli function 33
- vDSP_vfrac function 34
- vDSP_vfracD function 35
- vDSP_vgathr function 36
- vDSP_vgathra function 36
- vDSP_vgathraD function 37
- vDSP_vgathrD function 38
- vDSP_vgen function 39
- vDSP_vgenD function 39
- vDSP_vgenp function 40
- vDSP_vgenpD function 42
- vDSP_viclip function 43
- vDSP_viclipD function 44
- vDSP_vindex function 45
- vDSP_vindexD function 46
- vDSP_vlim function 47
- vDSP_vlimD function 47
- vDSP_vlint function 48
- vDSP_vlintD function 49
- vDSP_vnabs function 51
- vDSP_vnabsD function 51
- vDSP_vneg function 52
- vDSP_vnegD function 53
- vDSP_vramp function 53
- vDSP_vrampD function 54
- vDSP_vrsum function 55
- vDSP_vrsumD function 55
- vDSP_vrvrs function 56
- vDSP_vrvrsD function 57
- vDSP_vsimps function 57
- vDSP_vsimpsD function 58
- vDSP_vsort function 59
- vDSP_vsortD function 60
- vDSP_vsorti function 60
- vDSP_vsortiD function 61
- vDSP_vspdp function 62
- vDSP_vsq function 62
- vDSP_vsqD function 63
- vDSP_vssq function 63
- vDSP_vssqD function 64
- vDSP_vswsum function 64
- vDSP_vswsumD function 65
- vDSP_vtabi function 66
- vDSP_vtabiD function 67
- vDSP_vthr function 68
- vDSP_vthrD function 69
- vDSP_vthres function 70
- vDSP_vthresD function 70
- vDSP_vthrsc function 71
- vDSP_vthrscD function 72
- vDSP_vtrapz function 73
- vDSP_vtrapzD function 74
- vDSP_zvabs function 75
- vDSP_zvabsD function 75
- vDSP_zvconj function 76
- vDSP_zvconjD function 77
- vDSP_zvfill function 77

vDSP_zvfillD **function 78**
vDSP_zvmags **function 79**
vDSP_zvmagsD **function 79**
vDSP_zvmgsa **function 80**
vDSP_zvmgsaD **function 81**
vDSP_zvmov **function 82**
vDSP_zvmovD **function 82**
vDSP_zvneg **function 83**
vDSP_zvnegD **function 84**
vDSP_zvphas **function 85**
vDSP_zvphasD **function 85**