
vDSP Vector-to-Vector Arithmetic Operations Reference

[Performance > Carbon](#)



2009-01-06



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, Logic, 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 Vector-to-Vector Arithmetic Operations Reference 7

Overview	7
Functions by Task	7
Testing Bitwise Logical Equivalence	7
Doing Basic Arithmetic on Real Vectors	7
Doing Basic Arithmetic on Complex Vectors	9
Finding Maximum and Minimum Elements	10
Computing Vector Distance	10
Interpolating Between Two Vectors	10
Evaluating Vectors as Polynomials	11
Applying Pythagoras's Theorem to Vector Elements	11
Finding a Vector's Extrema	11
Swapping Elements Between Vectors	11
Merging Two Vectors	11
Computing Vector Spectra	12
Computing the Coherence Function of Two Vectors	12
Computing the Transfer Function	12
Doing Recursive Filtering on a Real Vector	12
Functions	12
vDSP_deq22	12
vDSP_deq22D	13
vDSP_vaam	14
vDSP_vaamD	15
vDSP_vadd	16
vDSP_vaddD	17
vDSP_vam	17
vDSP_vamD	18
vDSP_vasbm	18
vDSP_vasbmD	19
vDSP_vasm	21
vDSP_vasmD	21
vDSP_vdist	22
vDSP_vdistD	23
vDSP_vdiv	24
vDSP_vdivD	25
vDSP_vdivi	26
vDSP_venvlp	27
vDSP_venvlpD	29
vDSP_veqvi	30
vDSP_vintb	30
vDSP_vintbD	31

vDSP_vma 32
vDSP_vmaD 33
vDSP_vmax 34
vDSP_vmaxD 35
vDSP_vmaxmg 36
vDSP_vmaxmgD 37
vDSP_vmin 38
vDSP_vminD 39
vDSP_vminmg 40
vDSP_vminmgD 41
vDSP_vmma 42
vDSP_vmmaD 44
vDSP_vmmsb 45
vDSP_vmmsbD 46
vDSP_vmsa 47
vDSP_vmsaD 48
vDSP_vmsb 49
vDSP_vmsbD 50
vDSP_vmul 51
vDSP_vmulD 51
vDSP_vpoly 52
vDSP_vpolyD 53
vDSP_vpythg 54
vDSP_vpythgD 55
vDSP_vqint 57
vDSP_vqintD 58
vDSP_vsbm 59
vDSP_vsbmD 60
vDSP_vsbsbm 61
vDSP_vsbsbmD 62
vDSP_vsbsm 63
vDSP_vsbsmD 64
vDSP_vsub 65
vDSP_vsubD 65
vDSP_vswap 65
vDSP_vswapD 66
vDSP_vtmerg 67
vDSP_vtmergD 68
vDSP_zaspec 69
vDSP_zaspecD 69
vDSP_zcoher 70
vDSP_zcoherD 70
vDSP_zcspec 71
vDSP_zcspecD 72
vDSP_zrvadd 72
vDSP_zrvaddD 73

vDSP_zrdiv 74
vDSP_zrdivD 74
vDSP_zrmul 75
vDSP_zrmulD 75
vDSP_zrsub 76
vDSP_zrsubD 77
vDSP_ztrans 78
vDSP_ztransD 78
vDSP_zvadd 79
vDSP_zvaddD 80
vDSP_zvcma 80
vDSP_zvcmaD 81
vDSP_zvcmul 81
vDSP_zvcmulD 82
vDSP_zvmul 83
vDSP_zvmulD 84
vDSP_zvsub 84
vDSP_zvsubD 85

Document Revision History 87

Index 89

vDSP Vector-to-Vector Arithmetic Operations Reference

Framework:	Accelerate/vecLib
Declared in	vDSP.h

Overview

This document describes the C API for the vDSP functions that receive a vector as input and return a vector as output.

Functions by Task

Testing Bitwise Logical Equivalence

[vDSP_veqvi](#) (page 30)
Vector equivalence, 32-bit logical.

Doing Basic Arithmetic on Real Vectors

[vDSP_vadd](#) (page 16)
Adds vector A to vector B and leaves the result in vector C; single precision.

[vDSP_vaddD](#) (page 17)
Adds vector A to vector B and leaves the result in vector C; double precision.

[vDSP_vsub](#) (page 65)
Subtracts vector `signal1` from vector `signal2` and leaves the result in vector `result`; single precision.

[vDSP_vsubD](#) (page 65)
Subtracts vector `signal1` from vector `signal2` and leaves the result in vector `result`; double precision.

[vDSP_vam](#) (page 17)
Adds vectors A and B, multiplies the sum by vector C, and leaves the result in vector D; single precision.

[vDSP_vamD](#) (page 18)
Adds vectors A and B, multiplies the sum by vector C, and leaves the result in vector D; double precision.

[vDSP_vsbm](#) (page 59)
Vector subtract and multiply; single precision.

- vDSP_vsbmD (page 60)
Vector subtract and multiply; double precision.
- vDSP_vaam (page 14)
Vector add, add, and multiply; single precision.
- vDSP_vaamD (page 15)
Vector add, add, and multiply; double precision.
- vDSP_vsbsbm (page 61)
Vector subtract, subtract, and multiply; single precision.
- vDSP_vsbsbmD (page 62)
Vector subtract, subtract, and multiply; double precision.
- vDSP_vasbm (page 18)
Vector add, subtract, and multiply; single precision.
- vDSP_vasbmD (page 19)
Vector add, subtract, and multiply; double precision.
- vDSP_vasm (page 21)
Vector add and scalar multiply; single precision.
- vDSP_vasmD (page 21)
Vector add and scalar multiply; double precision.
- vDSP_vsbsm (page 63)
Vector subtract and scalar multiply; single precision.
- vDSP_vsbsmD (page 64)
Vector subtract and scalar multiply; double precision.
- vDSP_vmsa (page 47)
Vector multiply and scalar add; single precision.
- vDSP_vmsaD (page 48)
Vector multiply and scalar add; double precision.
- vDSP_vdiv (page 24)
Vector divide; single precision.
- vDSP_vdivD (page 25)
Vector divide; double precision.
- vDSP_vdivi (page 26)
Vector divide; integer.
- vDSP_vmul (page 51)
Multiplies vector A by vector B and leaves the result in vector C; single precision.
- vDSP_vmulD (page 51)
Multiplies vector A by vector B and leaves the result in vector C; double precision.
- vDSP_vma (page 32)
Vector multiply and add; single precision.
- vDSP_vmaD (page 33)
Vector multiply and add; double precision.
- vDSP_vmsb (page 49)
Vector multiply and subtract, single precision.
- vDSP_vmsbD (page 50)
Vector multiply and subtract; double precision.

[vDSP_vmma](#) (page 42)

Vector multiply, multiply, and add; single precision.

[vDSP_vmmaD](#) (page 44)

Vector multiply, multiply, and add; double precision.

[vDSP_vmsb](#) (page 45)

Vector multiply, multiply, and subtract; single precision.

[vDSP_vmsbD](#) (page 46)

Vector multiply, multiply, and subtract; double precision.

Doing Basic Arithmetic on Complex Vectors

[vDSP_zrdiv](#) (page 74)

Divides complex vector A by real vector B and leaves the result in vector C; single precision.

[vDSP_zrdivD](#) (page 74)

Divides complex vector A by real vector B and leaves the result in vector C; double precision.

[vDSP_zrvmul](#) (page 75)

Multiplies complex vector A by real vector B and leaves the result in vector C; single precision.

[vDSP_zrvmulD](#) (page 75)

Multiplies complex vector A by real vector B and leaves the result in vector C; double precision.

[vDSP_zrsub](#) (page 76)

Subtracts real vector B from complex vector A and leaves the result in complex vector C; single precision.

[vDSP_zrsubD](#) (page 77)

Subtracts real vector B from complex vector A and leaves the result in complex vector C; double precision.

[vDSP_zrvadd](#) (page 72)

Adds real vector B to complex vector A and leaves the result in complex vector C; single precision.

[vDSP_zrvaddD](#) (page 73)

Adds real vector B to complex vector A and leaves the result in complex vector C; double precision.

[vDSP_zvadd](#) (page 79)

Adds complex vectors A and B and leaves the result in complex vector C; single precision.

[vDSP_zvaddD](#) (page 80)

Adds complex vectors A and B and leaves the result in complex vector C; double precision.

[vDSP_zvcmul](#) (page 81)

Complex vector conjugate and multiply; single precision.

[vDSP_zvcmulD](#) (page 82)

Complex vector conjugate and multiply; double precision.

[vDSP_zvmul](#) (page 83)

Multiplies complex vectors A and B and leaves the result in complex vector C; single precision.

[vDSP_zvmulD](#) (page 84)

Multiplies complex vectors A and B and leaves the result in complex vector C; double precision.

[vDSP_zvsub](#) (page 84)

Subtracts complex vector B from complex vector A and leaves the result in complex vector C; single precision.

[vDSP_zvsubD](#) (page 85)

Subtracts complex vector B from complex vector A and leaves the result in complex vector C; double precision.

[vDSP_zvcma](#) (page 80)

Multiplies complex vector B by the complex conjugates of complex vector A, adds the products to complex vector C, and stores the results in complex vector D; single precision.

[vDSP_zvcmaD](#) (page 81)

Multiplies complex vector B by the complex conjugates of complex vector A, adds the products to complex vector C, and stores the results in complex vector D; double precision.

Finding Maximum and Minimum Elements

[vDSP_vmax](#) (page 34)

Vector maxima; single precision.

[vDSP_vmaxD](#) (page 35)

Vector maxima; double precision.

[vDSP_vmaxmg](#) (page 36)

Vector maximum magnitudes; single precision.

[vDSP_vmaxmgD](#) (page 37)

Vector maximum magnitudes; double precision.

[vDSP_vmin](#) (page 38)

Vector minima; single precision.

[vDSP_vminD](#) (page 39)

Vector minima; double precision.

[vDSP_vminmg](#) (page 40)

Vector minimum magnitudes; single precision.

[vDSP_vminmgD](#) (page 41)

Vector minimum magnitudes; double precision.

Computing Vector Distance

[vDSP_vdist](#) (page 22)

Vector distance; single precision.

[vDSP_vdistD](#) (page 23)

Vector distance; double precision.

Interpolating Between Two Vectors

[vDSP_vintb](#) (page 30)

Vector linear interpolation between vectors; single precision.

[vDSP_vintbD](#) (page 31)

Vector linear interpolation between vectors; double precision.

- [vDSP_vqint](#) (page 57)
Vector quadratic interpolation; single precision.
- [vDSP_vqintD](#) (page 58)
Vector quadratic interpolation; double precision.

Evaluating Vectors as Polynomials

- [vDSP_vpoly](#) (page 52)
Vector polynomial evaluation; single precision.
- [vDSP_vpolyD](#) (page 53)
Vector polynomial evaluation; double precision.

Applying Pythagoras's Theorem to Vector Elements

- [vDSP_vpythg](#) (page 54)
Vector pythagoras; single precision.
- [vDSP_vpythgD](#) (page 55)
Vector pythagoras; double precision.

Finding a Vector's Extrema

- [vDSP_venvlp](#) (page 27)
Vector envelope; single precision.
- [vDSP_venvlpD](#) (page 29)
Vector envelope; double precision.

Swapping Elements Between Vectors

- [vDSP_vswap](#) (page 65)
Vector swap; single precision.
- [vDSP_vswapD](#) (page 66)
Vector swap; double precision.

Merging Two Vectors

- [vDSP_vtmerng](#) (page 67)
Tapered merge of two vectors; single precision.
- [vDSP_vtmerngD](#) (page 68)
Tapered merge of two vectors; double precision.

Computing Vector Spectra

[vDSP_zaspec](#) (page 69)

Computes an accumulating autospectrum; single precision.

[vDSP_zaspecD](#) (page 69)

Computes an accumulating autospectrum; double precision.

[vDSP_zcspec](#) (page 71)

Accumulating cross-spectrum on two complex vectors; single precision.

[vDSP_zcspecD](#) (page 72)

Accumulating cross-spectrum on two complex vectors; double precision.

Computing the Coherence Function of Two Vectors

[vDSP_zcoher](#) (page 70)

Coherence function of two signals; single precision.

[vDSP_zcoherD](#) (page 70)

Coherence function of two signals; double precision.

Computing the Transfer Function

[vDSP_ztrans](#) (page 78)

Transfer function; single precision.

[vDSP_ztransD](#) (page 78)

Transfer function; double precision.

Doing Recursive Filtering on a Real Vector

[vDSP_deq22](#) (page 12)

Difference equation, 2 poles, 2 zeros; single precision.

[vDSP_deq22D](#) (page 13)

Difference equation, 2 poles, 2 zeros; double precision.

Functions

vDSP_deq22

Difference equation, 2 poles, 2 zeros; single precision.

```
void vDSP_deq22 (float * A,
                vDSP_Stride I,
                float * B,
                float * C,
                vDSP_Stride K,
                vDSP_Length N);
```

Parameters

<i>A</i>	Single-precision real input vector; must have at least N+2 elements
<i>I</i>	Stride for A
<i>B</i>	5 single-precision inputs, filter coefficients
<i>C</i>	Single-precision real output vector; must have at least N+2 elements
<i>K</i>	Stride for C
<i>N</i>	Number of new output elements to produce

Discussion

Performs two-pole two-zero recursive filtering on real input vector A. Since the computation is recursive, the first two elements in vector C must be initialized prior to calling vDSP_deq22. vDSP_deq22 creates N new values for vector C beginning with its third element and requires at least N+2 input values from vector A. This function can only be done out of place.

$$C_{nk} = \sum_{p=0}^2 A_{(n-p)i} B_p - \sum_{p=3}^4 C_{(n-p+2)k} B_p \quad n = \{2, N+1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_deq22D

Difference equation, 2 poles, 2 zeros; double precision.

```
void vDSP_deq22D (double * A,
                 vDSP_Stride I,
                 double * B,
                 double * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector; must have at least N+2 elements
----------	---

<i>I</i>	Stride for A
<i>B</i>	5 double-precision inputs, filter coefficients
<i>C</i>	Double-precision real output vector; must have at least N+2 elements
<i>K</i>	Stride for C
<i>N</i>	Number of new output elements to produce

Discussion

Performs two-pole two-zero recursive filtering on real input vector A. Since the computation is recursive, the first two elements in vector C must be initialized prior to calling `vDSP_deq22D`. `vDSP_deq22D` creates N new values for vector C beginning with its third element and requires at least N+2 input values from vector A. This function can only be done out of place.

$$C_{nk} = \sum_{p=0}^2 A_{(n-p)i} B_p - \sum_{p=3}^4 C_{(n-p+2)k} B_p \quad n = \{2, N+1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_vaam

Vector add, add, and multiply; single precision.

```
void vDSP_vaam (float * A,
               vDSP_Stride I,
               float * B,
               vDSP_Stride J,
               float * C,
               vDSP_Stride K,
               float * D,
               vDSP_Stride L,
               float * E,
               vDSP_Stride M,
               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 input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Single-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count ; each vector must have at least N elements

Discussion

This performs the operation

$$E_{nm} = (A_{ni} + B_{nj})(C_{nk} + D_{nl}) \quad n = \{0, N-1\}$$

Multiplies the sum of vectors A and B by the sum of vectors C and D. Results are stored in vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vaamD

Vector add, add, and multiply; double precision.

```
void vDSP_vaamD (double * A,
                vDSP_Stride I,
                double * B,
                vDSP_Stride J,
                double * C,
                vDSP_Stride K,
                double * D,
                vDSP_Stride L,
                double * E,
                vDSP_Stride M,
                vDSP_Length N);
```

Parameters

A
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 input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Double-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count ; each vector must have at least N elements

Discussion

This performs the operation

$$E_{nm} = (A_{ni} + B_{nj})(C_{nk} + D_{nl}) \quad n = \{0, N-1\}$$

Multiplies the sum of vectors A and B by the sum of vectors C and D. Results are stored in vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vadd

Adds vector A to vector B and leaves the result in vector C; single precision.

```
void vDSP_vadd (const float input1[],
                vDSP_Stride stride1,
                const float input2[],
                vDSP_Stride stride2,
                float result[],
                vDSP_Stride strideResult,
                vDSP_Length size);
```

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vaddD

Adds vector A to vector B and leaves the result in vector C; double precision.

```
void vDSP_vaddD (const double input1[],
                 vDSP_Stride stride1,
                 const double input2[],
                 vDSP_Stride stride2,
                 double result[],
                 vDSP_Stride strideResult,
                 vDSP_Length size);
```

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vam

Adds vectors A and B, multiplies the sum by vector C, and leaves the result in vector D; single precision.

```
void vDSP_vam (const float input1[],
               vDSP_Stride stride1,
               const float input2[],
               vDSP_Stride stride2,
               const float input3[],
               vDSP_Stride stride3,
               float result[],
               vDSP_Stride strideResult,
               vDSP_Length size);
```

Discussion

This performs the operation

$$D_{nL} = (A_{nI} + B_{nJ}) C_{nK} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vamD

Adds vectors A and B, multiplies the sum by vector C, and leaves the result in vector D; double precision.

```
void vDSP_vamD (const double input1[],
               vDSP_Stride stride1,
               const double input2[],
               vDSP_Stride stride2,
               const double input3[],
               vDSP_Stride stride3,
               double result[],
               vDSP_Stride strideResult,
               vDSP_Length size);
```

Discussion

This performs the operation

$$D_{nL} = (A_{nI} + B_{nJ})C_{nK} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vasbm

Vector add, subtract, and multiply; single precision.

```
void vDSP_vasbm (float * A,
                vDSP_Stride I,
                float * B,
                vDSP_Stride J,
                float * C,
                vDSP_Stride K,
                float * D,
                vDSP_Stride L,
                float * E,
                vDSP_Stride M,
                vDSP_Length N);
```

Parameters

A
Single-precision real input vector

I
Stride for A

<i>B</i>	Single-precision real input vector
<i>J</i>	Stride for B
<i>C</i>	Single-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Single-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count ; each vector must have at least N elements

Discussion

This performs the operation

$$E_{nM} = (A_{nI} + B_{nJ})(C_{nK} - D_{nL}), \quad n = \{0, N-1\}$$

Multiplies the sum of vectors A and B by the result of subtracting vector D from vector C. Results are stored in vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vasbmD

Vector add, subtract, and multiply; double precision.

```
void vDSP_vasbmD (double * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
double * D,
vDSP_Stride L,
double * E,
vDSP_Stride M,
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 input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Double-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count ; each vector must have at least N elements

Discussion

This performs the operation

$$E_{nM} = (A_{nI} + B_{nJ})(C_{nK} - D_{nL}), \quad n = \{0, N-1\}$$

Multiplies the sum of vectors A and B by the result of subtracting vector D from vector C. Results are stored in vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vasm

Vector add and scalar multiply; single precision.

```
void vDSP_vasm (float * A,
                vDSP_Stride I,
                float * B,
                vDSP_Stride J,
                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 vector
<i>J</i>	Stride for B
<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 ; each vector must have at least N elements

Discussion

This performs the operation

$$D_{nM} = (A_{nI} + B_{nJ}) C, \quad n = \{0, N-1\}$$

Multiplies the sum of vectors A and B by scalar C. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vasmD

Vector add and scalar multiply; double precision.

```
void vDSP_vasmD (double * A,
                vDSP_Stride I,
                double * B,
                vDSP_Stride J,
                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 vector
<i>J</i>	Stride for B
<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 ; each vector must have at least N elements

Discussion

This performs the operation

$$D_{nM} = (A_{nI} + B_{nJ})C, \quad n = \{0, N-1\}$$

Multiplies the sum of vectors A and B by scalar C. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdist

Vector distance; single precision.

```
void vDSP_vdist (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

$$C_{nk} = \sqrt{A_{ni}^2 + B_{nj}^2} \quad n = \{0, N-1\}$$

Computes the square root of the sum of the squares of corresponding elements of vectors A and B, and stores the result in the corresponding element of vector C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdistD

Vector distance; double precision.

```
void vDSP_vdistD (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

$$C_{nk} = \sqrt{A_{ni}^2 + B_{nj}^2} \quad n = \{0, N-1\}$$

Computes the square root of the sum of the squares of corresponding elements of vectors A and B, and stores the result in the corresponding element of vector C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdiv

Vector divide; single precision.


```
void vDSP_vdiv (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

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

Divides elements of vector A by corresponding elements of vector B, and stores the results in corresponding elements of vector C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdivD

Vector divide; double precision.

```
void vDSP_vdivD (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

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

Divides elements of vector A by corresponding elements of vector B, and stores the results in corresponding elements of vector C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vdivi

Vector divide; integer.

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

Parameters

<i>A</i>	Integer input vector
<i>I</i>	Stride for A
<i>B</i>	Integer input vector
<i>J</i>	Stride for B
<i>C</i>	Integer output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

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

Divides elements of vector *A* by corresponding elements of vector *B*, and stores the results in corresponding elements of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_venvlp

Vector envelope; single precision.

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

Parameters

<i>A</i>	Single-precision real input vector: high envelope
<i>I</i>	Stride for A
<i>B</i>	Single-precision real input vector: low envelope
<i>J</i>	Stride for B
<i>C</i>	Single-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

$$\text{If } C_{nK} > A_{nI} \text{ or } C_{nK} < B_{nJ} \text{ then } D_{nM} = C_{nK}$$

$$\text{else } D_{nM} = 0.0 \quad n = \{0, N-1\}$$

Finds the extrema of vector C. For each element of C, the corresponding element of A provides an upper-threshold value, and the corresponding element of B provides a lower-threshold value. If the value of an element of C falls outside the range defined by these thresholds, it is copied to the corresponding element of vector D. If its value is within the range, the corresponding element of vector D is set to zero.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_venvlpD

Vector envelope; double precision.

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

Parameters

<i>A</i>	Double-precision real input vector: high envelope
<i>I</i>	Stride for A
<i>B</i>	Double-precision real input vector: low envelope
<i>J</i>	Stride for B
<i>C</i>	Double-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

$$\text{If } C_{nK} > A_{nI} \text{ or } C_{nK} < B_{nJ} \text{ then } D_{nM} = C_{nK}$$

$$\text{else } D_{nM} = 0.0 \quad n = \{0, N-1\}$$

Finds the extrema of vector C. For each element of C, the corresponding element of A provides an upper-threshold value, and the corresponding element of B provides a lower-threshold value. If the value of an element of C falls outside the range defined by these thresholds, it is copied to the corresponding element of vector D. If its value is within the range, the corresponding element of vector D is set to zero.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_veqvi

Vector equivalence, 32-bit logical.

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

Parameters

<i>A</i>	Integer input vector
<i>I</i>	Stride for A
<i>B</i>	Integer input vector
<i>J</i>	Stride for B
<i>C</i>	Integer output vector
<i>K</i>	Stride for C
<i>N</i>	Count

Discussion

Performs the operation

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

Outputs the bitwise logical equivalence, exclusive NOR, of the integers of vectors A and B. For each pair of input values, bits in each position are compared. A bit in the output value is set if both input bits are set, or both are clear; otherwise it is cleared.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vintb

Vector linear interpolation between vectors; single precision.

```
void vDSP_vintb (float * A,
                vDSP_Stride I,
                float * B,
                vDSP_Stride J,
                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 vector
<i>J</i>	Stride for B
<i>C</i>	Single-precision real input scalar: interpolation constant
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

$$D_{nK} = A_{nI} + C[B_{nJ} - A_{nI}] \quad n = \{0, N-1\}$$

Creates vector D by interpolating between vectors A and B.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vintbD

Vector linear interpolation between vectors; double precision.

```
void vDSP_vintbD (double * A,
                 vDSP_Stride I,
                 double * B,
                 vDSP_Stride J,
                 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 vector
<i>J</i>	Stride for B
<i>C</i>	Double-precision real input scalar: interpolation constant
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

$$D_{nK} = A_{nI} + C[B_{nJ} - A_{nI}] \quad n = \{0, N-1\}$$

Creates vector D by interpolating between vectors A and B.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vma

Vector multiply and add; single precision.


```
void vDSP_vma (float * A,
              vDSP_Stride I,
              float * B,
              vDSP_Stride J,
              float * C,
              vDSP_Stride K,
              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 vector
<i>J</i>	Stride for B
<i>C</i>	Single-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

This performs the operation

$$D_{nM} = A_{nI} \cdot B_{nJ} + C_{nK} \quad n = \{0, N-1\}$$

Multiplies corresponding elements of vectors A and B, add the corresponding elements of vector C, and stores the results in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmaD

Vector multiply and add; double precision.

```
void vDSP_vmaD (double * A,
                vDSP_Stride I,
                double * B,
                vDSP_Stride J,
                double * C,
                vDSP_Stride K,
                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 vector
<i>J</i>	Stride for B
<i>C</i>	Double-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

This performs the operation

$$D_{nM} = A_{nI} \cdot B_{nJ} + C_{nK} \quad n = \{0, N-1\}$$

Multiplies corresponding elements of vectors A and B, add the corresponding elements of vector C, and stores the results in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmax

Vector maxima; single precision.

```
void vDSP_vmax (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

This performs the operation

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

Each element of output vector *C* is the greater of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmaxD

Vector maxima; double precision.

```
void vDSP_vmaxD (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

This performs the operation

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

Each element of output vector *C* is the greater of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmaxmg

Vector maximum magnitudes; single precision.

```
void vDSP_vmaxmg (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

This performs the operation

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

Each element of output vector *C* is the larger of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmaxmgD

Vector maximum magnitudes; double precision.

```
void vDSP_vmaxmgD (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

This performs the operation

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

Each element of output vector *C* is the larger of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmin

Vector minima; single precision.

```
void vDSP_vmin (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

This performs the operation

If $A_{nI} \leq B_{nJ}$ then $C_{nK} = A_{nI}$ else $C_{nK} = B_{nJ}$ $n = \{0, N-1\}$

Each element of output vector *C* is the lesser of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vminD

Vector minima; double precision.

```
void vDSP_vminD (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

This performs the operation

If $A_{nI} \leq B_{nJ}$ then $C_{nK} = A_{nI}$ else $C_{nK} = B_{nJ}$ $n = \{0, N-1\}$

Each element of output vector *C* is the lesser of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vminmg

Vector minimum magnitudes; single precision.


```
void vDSP_vmin (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

This performs the operation

$$\text{If } |A_{nI}| \leq |B_{nJ}| \text{ then } C_{nK} = |A_{nI}| \text{ else } C_{nK} = |B_{nJ}| \quad n = \{0, N-1\}$$

Each element of output vector *C* is the smaller of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vminmgD

Vector minimum magnitudes; double precision.

```
void vDSP_vminD (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

This performs the operation

$$\text{If } |A_{nI}| \leq |B_{nJ}| \text{ then } C_{nK} = |A_{nI}| \text{ else } C_{nK} = |B_{nJ}| \quad n = \{0, N-1\}$$

Each element of output vector *C* is the smaller of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmma

Vector multiply, multiply, and add; single precision.

```
void vDSP_vmma (float * A,
vDSP_Stride I,
float * B,
vDSP_Stride J,
float * C,
vDSP_Stride K,
float * D,
vDSP_Stride L,
float * E,
vDSP_Stride M,
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 input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Single-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} \cdot B_{nJ} + C_{nK} \cdot D_{nL} \quad n = \{0, N-1\}$$

Corresponding elements of A and B are multiplied, corresponding values of C and D are multiplied, and these products are added together and stored in E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmmaD

Vector multiply, multiply, and add; double precision.

```
void vDSP_vmmaD (double * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
double * D,
vDSP_Stride L,
double * E,
vDSP_Stride M,
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 input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Double-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} \cdot B_{nJ} + C_{nK} \cdot D_{nL} \quad n = \{0, N-1\}$$

Corresponding elements of A and B are multiplied, corresponding values of C and D are multiplied, and these products are added together and stored in E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmmsb

Vector multiply, multiply, and subtract; single precision.

```
void vDSP_vmmsb (float * A,
                vDSP_Stride I,
                float * B,
                vDSP_Stride J,
                float * C,
                vDSP_Stride K,
                float * D,
                vDSP_Stride L,
                float * E,
                vDSP_Stride M,
                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 input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Single-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} B_{nJ} - C_{nK} D_{nL} \quad n = \{0, N-1\}$$

Corresponding elements of *A* and *B* are multiplied, corresponding values of *C* and *D* are multiplied, and the second product is subtracted from the first. The result is stored in *E*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmmsbD

Vector multiply, multiply, and subtract; double precision.

```
void vDSP_vmmsbD (double * A,
                 vDSP_Stride I,
                 double * B,
                 vDSP_Stride J,
                 double * C,
                 vDSP_Stride K,
                 double * D,
                 vDSP_Stride L,
                 double * E,
                 vDSP_Stride M,
                 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 vector
<i>J</i>	Stride for <i>B</i>
<i>C</i>	Double-precision real input vector
<i>K</i>	Stride for <i>C</i>
<i>D</i>	Double-precision real input vector
<i>L</i>	Stride for <i>D</i>
<i>E</i>	Double-precision real output vector
<i>M</i>	Stride for <i>E</i>
<i>N</i>	Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} B_{nJ} - C_{nK} D_{nL} \quad n = \{0, N-1\}$$

Corresponding elements of A and B are multiplied, corresponding values of C and D are multiplied, and the second product is subtracted from the first. The result is stored in E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmsa

Vector multiply and scalar add; single precision.

```
void vDSP_vmsa (float * A,
               vDSP_Stride I,
               float * B,
               vDSP_Stride J,
               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 vector
<i>J</i>	Stride for B
<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

This performs the operation

$$D_{nK} = A_{nI} \cdot B_{nJ} + C \quad n = \{0, N-1\}$$

Corresponding elements of *A* and *B* are multiplied and the scalar *C* is added. The result is stored in *D*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmsaD

Vector multiply and scalar add; double precision.

```
void vDSP_vmsaD (double * A,
                vDSP_Stride I,
                double * B,
                vDSP_Stride J,
                double * C,
                double * D,
                vDSP_Stride L,
                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 vector
<i>J</i>	Stride for <i>B</i>
<i>C</i>	Double-precision real input scalar
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for <i>D</i>
<i>N</i>	Count

Discussion

This performs the operation

$$D_{nK} = A_{nI} \cdot B_{nJ} + C \quad n = \{0, N-1\}$$

Corresponding elements of *A* and *B* are multiplied and the scalar *C* is added. The result is stored in *D*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmsb

Vector multiply and subtract, single precision.

```
void vDSP_vmsb (float * A,
               vDSP_Stride I,
               float * B,
               vDSP_Stride J,
               float * C,
               vDSP_Stride K,
               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 vector
<i>J</i>	Stride for B
<i>C</i>	Single-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

This performs the operation

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

Corresponding elements of A and B are multiplied and the corresponding value of C is subtracted. The result is stored in D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmsbD

Vector multiply and subtract; double precision.

```
void vDSP_vmsbD (double * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
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 vector
<i>J</i>	Stride for B
<i>C</i>	Double-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

This performs the operation

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

Corresponding elements of A and B are multiplied and the corresponding value of C is subtracted. The result is stored in D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmul

Multiplies vector A by vector B and leaves the result in vector C; single precision.

```
void vDSP_vmul (const float A[],
               vDSP_Stride I,
               const float B[],
               vDSP_Stride J,
               float C[],
               vDSP_Stride K,
               vDSP_Length N);
```

Parameters

<i>A</i>	Input vector
<i>I</i>	Address stride for A
<i>B</i>	Input vector
<i>J</i>	Address stride for B
<i>C</i>	Output vector
<i>K</i>	Address stride for C
<i>N</i>	Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vmulD

Multiplies vector A by vector B and leaves the result in vector C; double precision.

```
void vDSP_vmulD (const double A[],
                 vDSP_Stride I,
                 const double B[],
                 vDSP_Stride J,
                 double C[],
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Input vector
<i>I</i>	Address stride for A
<i>B</i>	Input vector
<i>J</i>	Address stride for B
<i>C</i>	Output vector
<i>K</i>	Address stride for C
<i>N</i>	Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vpoly

Vector polynomial evaluation; single precision.

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

Parameters

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

<i>I</i>	Stride for A
<i>B</i>	Single-precision real input vector: variable values
<i>J</i>	Stride for B
<i>C</i>	Single-precision real output vector
<i>K</i>	Stride for C
<i>N</i>	Count
<i>P</i>	Degree of polynomial

Discussion

Performs the operation

$$C_{nK} = \sum_{p=0}^P A_{pI} \cdot B_{nJ}^{P-p} \quad n = \{0, N-1\}$$

Evaluates polynomials using vector B as independent variables and vector A as coefficients. A polynomial of degree p requires p+1 coefficients, so vector A should contain P+1 values.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vpolyD

Vector polynomial evaluation; double precision.

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

Parameters

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

<i>B</i>	Double-precision real input vector: variable values
<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
<i>P</i>	Degree of polynomial

Discussion

Performs the operation

$$C_{nK} = \sum_{p=0}^P A_{pI} \cdot B_{nJ}^{P-p} \quad n = \{0, N-1\}$$

Evaluates polynomials using vector *B* as independent variables and vector *A* as coefficients. A polynomial of degree *p* requires *p*+1 coefficients, so vector *A* should contain *P*+1 values.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vpythg

Vector pythagoras; single precision.

```
void vDSP_vpythg (float * A,
                 vDSP_Stride I,
                 float * B,
                 vDSP_Stride J,
                 float * C,
                 vDSP_Stride K,
                 float * D,
                 vDSP_Stride L,
                 float * E,
                 vDSP_Stride M,
                 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 vector
<i>J</i>	Stride for B
<i>C</i>	Single-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Single-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count

Discussion

Performs the operation

$$E_{nM} = \sqrt{(A_{nI} - C_{nK})^2 + (B_{nJ} - D_{nL})^2} \quad n = \{0, N-1\}$$

Subtracts vector C from A and squares the differences, subtracts vector D from B and squares the differences, adds the two sets of squared differences, and then writes the square roots of the sums to vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vpythgD

Vector pythagoras; double precision.

```
void vDSP_vpythgD (double * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
double * D,
vDSP_Stride L,
double * E,
vDSP_Stride M,
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 input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Double-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count

Discussion

Performs the operation

$$E_{nM} = \sqrt{(A_{nI} - C_{nK})^2 + (B_{nJ} - D_{nL})^2} \quad n = \{0, N-1\}$$

Subtracts vector C from A and squares the differences, subtracts vector D from B and squares the differences, adds the two sets of squared differences, and then writes the square roots of the sums to vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vqint

Vector quadratic interpolation; single precision.

```
void vDSP_vqint (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 <i>A</i> and fractional parts are interpolation constants
<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 for <i>C</i>
<i>M</i>	Length of <i>A</i> : must be greater than or equal to 3

Discussion

Performs the operation

$$C_{nK} = \frac{A_{\beta-1}[\alpha^2 - \alpha] + A_{\beta}[2.0 - 2.0\alpha^2] + A_{\beta+1}[\alpha^2 + \alpha]}{2}$$

where: $\beta = \max(\text{trunc}(B_{nJ}), 1)$ $n = \{0, N-1\}$

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

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 triple of adjacent values in vector *A*.

The value of the corresponding element of *C* is derived from these three values by quadratic 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 less than or equal to *M* - 2.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vqintD

Vector quadratic interpolation; double precision.

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

Parameters*A*

Double-precision real input vector

*B*Double-precision real input vector: integer parts are indices into *A* and fractional parts are interpolation constants*J*Stride for *B**C*

Double-precision real output vector

*K*Stride for *C**N*Count for *C**M*Length of *A*: must be greater than or equal to 3**Discussion**

Performs the operation

$$C_{nK} = \frac{A_{\beta-1}[\alpha^2 - \alpha] + A_{\beta}[2.0 - 2.0\alpha^2] + A_{\beta+1}[\alpha^2 + \alpha]}{2}$$

where: $\beta = \max(\text{trunc}(B_{nJ}), 1)$ $n = \{0, N-1\}$ $\alpha = B_{nJ} - \text{float}(\beta)$

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 triple of adjacent values in vector *A*.

The value of the corresponding element of *C* is derived from these three values by quadratic 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 less than or equal to *M* - 2.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsbm

Vector subtract and multiply; single precision.

```
void vDSP_vsbm (float * A,
               vDSP_Stride I,
               float * B,
               vDSP_Stride J,
               float * C,
               vDSP_Stride K,
               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 vector
<i>J</i>	Stride for B
<i>C</i>	Single-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

$$D_{nM} = (A_{nI} - B_{nJ}) C_{nK} \quad n = \{0, N-1\}$$

Subtracts vector B from vector A and then multiplies the differences by vector C. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsbdM

Vector subtract and multiply; double precision.

```
void vDSP_vsbdM (double * A,
                vDSP_Stride I,
                double * B,
                vDSP_Stride J,
                double * C,
                vDSP_Stride K,
                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 vector
<i>J</i>	Double for B
<i>C</i>	Double-precision real input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real output vector
<i>L</i>	Stride for D
<i>N</i>	Count

Discussion

Performs the operation

$$D_{nM} = (A_{nI} - B_{nJ}) C_{nK} \quad n = \{0, N-1\}$$

Subtracts vector B from vector A and then multiplies the differences by vector C. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsbsbm

Vector subtract, subtract, and multiply; single precision.

```
void vDSP_vsbsbm (float * A,
                 vDSP_Stride I,
                 float * B,
                 vDSP_Stride J,
                 float * C,
                 vDSP_Stride K,
                 float * D,
                 vDSP_Stride L,
                 float * E,
                 vDSP_Stride M,
                 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 input vector
<i>K</i>	Stride for C
<i>D</i>	Single-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Single-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count

Discussion

Performs the operation

$$E_{nM} = (A_{nI} - B_{nJ})(C_{nK} - D_{nL}) \quad n = \{0, N-1\}$$

Subtracts vector B from A, subtracts vector D from C, and multiplies the differences. Results are stored in vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsbsbmD

Vector subtract, subtract, and multiply; double precision.

```
void vDSP_vsbsbmD (double * A,
vDSP_Stride I,
double * B,
vDSP_Stride J,
double * C,
vDSP_Stride K,
double * D,
vDSP_Stride L,
double * E,
vDSP_Stride M,
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 input vector
<i>K</i>	Stride for C
<i>D</i>	Double-precision real input vector
<i>L</i>	Stride for D
<i>E</i>	Double-precision real output vector
<i>M</i>	Stride for E
<i>N</i>	Count

Discussion

Performs the operation

$$E_{nM} = (A_{nI} - B_{nJ})(C_{nK} - D_{nL}) \quad n = \{0, N-1\}$$

Subtracts vector B from A, subtracts vector D from C, and multiplies the differences. Results are stored in vector E.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsbsm

Vector subtract and scalar multiply; single precision.

```
void vDSP_vsbsm (float * A,
                vDSP_Stride I,
                float * B,
                vDSP_Stride J,
                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 vector
<i>J</i>	Stride for B
<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

$$D_{nK} = (A_{nI} - B_{nJ})C \quad n = \{0, N-1\}$$

Subtracts vector B from vector A and then multiplies each difference by scalar C. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsbsmD

Vector subtract and scalar multiply; double precision.

```
void vDSP_vsbsmD (double * A,
                 vDSP_Stride I,
                 double * B,
                 vDSP_Stride J,
                 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 vector
<i>J</i>	Stride for B
<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

Performs the operation

$$D_{nK} = (A_{nI} - B_{nJ})C \quad n = \{0, N-1\}$$

Subtracts vector B from vector A and then multiplies each difference by scalar C. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsub

Subtracts vector `signal1` from vector `signal2` and leaves the result in vector `result`; single precision.

```
void vDSP_vsub (const float input1[],
               vDSP_Stride stride1,
               const float input2[],
               vDSP_Stride stride2,
               float result[],
               vDSP_Stride strideResult,
               vDSP_Length size);
```

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsubD

Subtracts vector `signal1` from vector `signal2` and leaves the result in vector `result`; double precision.

```
void vDSP_vsub (const float input1[],
               vDSP_Stride stride1,
               const float input2[],
               vDSP_Stride stride2,
               float result[],
               vDSP_Stride strideResult,
               vDSP_Length size);
```

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vswap

Vector swap; single precision.

```
void vDSP_vswap (float * A,
                vDSP_Stride I,
                float * B,
                vDSP_Stride J,
                vDSP_Length N);
```

Parameters

A
Single-precision real input-output vector

I
Stride for A

B
Single-precision real input-output vector

J
Stride for B

N
Count

Discussion

Performs the operation

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

Exchanges the elements of vectors A and B.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vswapD

Vector swap; double precision.

```
void vDSP_vswapD (double * A,
                 vDSP_Stride I,
                 double * B,
                 vDSP_Stride J,
                 vDSP_Length N);
```

Parameters

A
Double-precision real input-output vector

I
Stride for A

B
Double-precision real input-output vector

J
Stride for B

N

Count

Discussion

Performs the operation

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

Exchanges the elements of vectors A and B.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vtmerge

Tapered merge of two vectors; single precision.

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

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

Discussion

Performs the operation

$$C_{nK} = A_{nI} + \frac{n(B_{nJ} - A_{nI})}{N-1} \quad n = \{0, N-1\}$$

Performs a tapered merge of vectors A and B. Values written to vector C range from element zero of vector A to element N-1 of vector B. Output values between these endpoints reflect varying amounts of their corresponding inputs from vectors A and B, with the percentage of vector A decreasing and the percentage of vector B increasing as the index increases.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vtmergD

Tapered merge of two vectors; double precision.

```
void vDSP_vtmergD (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

$$C_{nK} = A_{nI} + \frac{n(B_{nJ} - A_{nI})}{N-1} \quad n = \{0, N-1\}$$

Performs a tapered merge of vectors A and B. Values written to vector C range from element zero of vector A to element N-1 of vector B. Output values between these endpoints reflect varying amounts of their corresponding inputs from vectors A and B, with the percentage of vector A decreasing and the percentage of vector B increasing as the index increases.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zaspec

Computes an accumulating autospectrum; single precision.

```
void vDSP_zaspec (DSPSplitComplex * A,
float * C,
vDSP_Length N);
```

Parameters

A
Input vector

C
Input-output vector

N
Real output count

Discussion

vDSP_zaspec multiplies single-precision complex vector *A* by its complex conjugates, yielding the sums of the squares of the complex and real parts: $(x + iy)(x - iy) = (x^2 + y^2)$. The results are added to real single-precision input-output vector *C*. Vector *C* must contain valid data from previous processing or should be initialized according to your needs before calling vDSP_zaspec.

$$C_n = C_n + (Re(A_n))^2 + (Im(A_n))^2 \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zaspecD

Computes an accumulating autospectrum; double precision.

```
void vDSP_zaspecD (DSPDoubleSplitComplex * A,
double * C,
vDSP_Length N);
```

Parameters

A
Input vector

C
Input-output vector

N

Real output count

Discussion

`vDSP_zaspecD` multiplies double-precision complex vector A by its complex conjugates, yielding the sums of the squares of the complex and real parts: $(x + iy)(x - iy) = (x^2 + y^2)$. The results are added to real double-precision input-output vector C . Vector C must contain valid data from previous processing or should be initialized according to your needs before calling `vDSP_zaspec`.

$$C_n = C_n + (Re(A_n))^2 + (Im(A_n))^2 \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zcoher

Coherence function of two signals; single precision.

```
void vDSP_zcoher (float * A,
                 float * B,
                 DSPSplitComplex * C,
                 float * D,
                 vDSP_Length N);
```

Discussion

Computes the single-precision coherence function D of two signals. The inputs are the signals' autospectra, real single-precision vectors A and B , and their cross-spectrum, single-precision complex vector C .

$$D_n = \frac{[Re(C_n)]^2 + [Im(C_n)]^2}{A_n B_n} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zcoherD

Coherence function of two signals; double precision.

```
void vDSP_zcoherD (double * A,
double * B,
DSPDoubleSplitComplex * C,
double * D,
vDSP_Length N);
```

Discussion

Computes the double-precision coherence function D of two signals. The inputs are the signals' autospectra, real double-precision vectors A and B , and their cross-spectrum, double-precision complex vector C .

$$D_n = \frac{[\operatorname{Re}(C_n)]^2 + [\operatorname{Im}(C_n)]^2}{A_n B_n} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zcspec

Accumulating cross-spectrum on two complex vectors; single precision.

```
void vDSP_zcspec (DSPSplitComplex * A,
DSPSplitComplex * B,
DSPSplitComplex * C,
vDSP_Length N);
```

Parameters

A	Single-precision complex input vector
B	Single-precision complex input vector
C	Single-precision complex input-output vector
N	Count

Discussion

Computes the cross-spectrum of complex vectors A and B and then adds the results to complex input-output vector C . Vector C should contain valid data from previous processing or should be initialized with zeros before calling `vDSP_zcspec`.

$$C_n = C_n + A_n^* B_n \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zcspecD

Accumulating cross-spectrum on two complex vectors; double precision.

```
void vDSP_zcspecD (DSPDoubleSplitComplex * A,
                  DSPDoubleSplitComplex * B,
                  DSPDoubleSplitComplex * C,
                  vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision complex input vector
<i>B</i>	Double-precision complex input vector
<i>C</i>	Double-precision complex input-output vector
<i>N</i>	Count

Discussion

Computes the cross-spectrum of complex vectors *A* and *B* and then adds the results to complex input-output vector *C*. Vector *C* should contain valid data from previous processing or should be initialized with zeros before calling `vDSP_zcspecD`.

$$C_n = C_n + A_n^* B_n \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrvadd

Adds real vector *B* to complex vector *A* and leaves the result in complex vector *C*; single precision.

```
void vDSP_zrvadd (DSPSplitComplex * A,
                 vDSP_Stride I,
                 const float B[],
                 vDSP_Stride J,
                 DSPSplitComplex * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Input vector
<i>I</i>	Address stride for <i>A</i>
<i>B</i>	Input vector

J
Address stride for B

C
Output vector

K
Address stride for C

N
Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrvaddD

Adds real vector B to complex vector A and leaves the result in complex vector C; double precision.

```
void vDSP_zrvaddD (DSPDoubleSplitComplex * A,
                  vDSP_Stride I,
                  const double B[],
                  vDSP_Stride J,
                  DSPDoubleSplitComplex * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

A
Input vector

I
Address stride for A

B
Input vector

J
Address stride for B

C
Output vector

K
Address stride for C

N
Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrdiv

Divides complex vector A by real vector B and leaves the result in vector C; single precision.

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

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrdivD

Divides complex vector A by real vector B and leaves the result in vector C; double precision.

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

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrvmul

Multiplies complex vector A by real vector B and leaves the result in vector C; single precision.

```
void vDSP_zrvmul (DSPSplitComplex * A,
                 vDSP_Stride I,
                 const float B[],
                 vDSP_Stride J,
                 DSPSplitComplex * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Input vector
<i>I</i>	Address stride for A
<i>B</i>	Input vector
<i>J</i>	Address stride for B
<i>C</i>	Output vector
<i>K</i>	Address stride for C
<i>N</i>	Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrvmulD

Multiplies complex vector A by real vector B and leaves the result in vector C; double precision.

```
void vDSP_zrvmulD (DSPDoubleSplitComplex * A,
                  vDSP_Stride I,
                  const double B[],
                  vDSP_Stride J,
                  DSPDoubleSplitComplex * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

<i>A</i>	Input vector
<i>I</i>	Address stride for A
<i>B</i>	Input vector
<i>J</i>	Address stride for B
<i>C</i>	Output vector
<i>K</i>	Address stride for C
<i>N</i>	Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrvsub

Subtracts real vector B from complex vector A and leaves the result in complex vector C; single precision.

```
void vDSP_zrvsub (DSPSplitComplex * A,
                  vDSP_Stride I,
                  const float B[],
                  vDSP_Stride J,
                  DSPSplitComplex * C,
                  vDSP_Stride K,
                  vDSP_Length N);
```

Parameters

<i>A</i>	Input vector
----------	--------------

<i>I</i>	Address stride for A
<i>B</i>	Input vector
<i>J</i>	Address stride for B
<i>C</i>	Output vector
<i>K</i>	Address stride for C
<i>N</i>	Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrsubD

Subtracts real vector B from complex vector A and leaves the result in complex vector C; double precision.

```
void vDSP_zrsubD (DSPDoubleSplitComplex * A,
                 vDSP_Stride I,
                 const double B[],
                 vDSP_Stride J,
                 DSPDoubleSplitComplex * C,
                 vDSP_Stride K,
                 vDSP_Length N);
```

Parameters

<i>A</i>	Input vector
<i>I</i>	Address stride for A
<i>B</i>	Input vector
<i>J</i>	Address stride for B
<i>C</i>	Output vector
<i>K</i>	Address stride for C

N

Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_ztrans

Transfer function; single precision.

```
void vDSP_ztrans (float * A,
                 DSPSplitComplex * B,
                 DSPSplitComplex * C,
                 vDSP_Length N);
```

Parameters A

Single-precision real input vector

 B

Single-precision complex input vector

 C

Single-precision complex output vector

 N

Count

Discussion

This performs the operation

$$C_n = \frac{B_n}{A_n} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_ztransD

Transfer function; double precision.

```
void vDSP_ztransD (double * A,
                  DSPDoubleSplitComplex * B,
                  DSPDoubleSplitComplex * C,
                  vDSP_Length N);
```

Parameters

<i>A</i>	Double-precision real input vector
<i>B</i>	Double-precision complex input vector
<i>C</i>	Double-precision complex output vector
<i>N</i>	Count

Discussion

This performs the operation

$$C_n = \frac{B_n}{A_n} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvadd

Adds complex vectors *A* and *B* and leaves the result in complex vector *C*; single precision.

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

Parameters

<i>A</i>	Input vector
<i>I</i>	Address stride for <i>A</i>
<i>B</i>	Input vector
<i>J</i>	Address stride for <i>B</i>
<i>C</i>	Output vector

K

Address stride for C

 N

Complex output count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvaddD

Adds complex vectors A and B and leaves the result in complex vector C; double precision.

```
void vDSP_zvaddD (DSPDoubleSplitComplex * input1,
                 vDSP_Stride stride1,
                 DSPDoubleSplitComplex * input2,
                 vDSP_Stride stride2,
                 DSPDoubleSplitComplex * result,
                 vDSP_Stride strideResult,
                 vDSP_Length size);
```

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvcma

Multiplies complex vector B by the complex conjugates of complex vector A, adds the products to complex vector C, and stores the results in complex vector D; single precision.


```
void vDSP_zvcma (const DSPSplitComplex * input1,
                vDSP_Stride stride1,
                const DSPSplitComplex * input2,
                vDSP_Stride stride2,
                DSPSplitComplex * input3,
                vDSP_Stride stride3,
                DSPSplitComplex * result,
                vDSP_Stride strideResult,
                vDSP_Length size);
```

Discussion

This performs the operation

$$D_{nL} = A_{nI}^* B_{nJ} + C_{nK} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvcmaD

Multiplies complex vector B by the complex conjugates of complex vector A, adds the products to complex vector C, and stores the results in complex vector D; double precision.

```
void vDSP_zvcmaD (DSPDoubleSplitComplex * input1,
                 vDSP_Stride stride1,
                 DSPDoubleSplitComplex * input2,
                 vDSP_Stride stride2,
                 DSPDoubleSplitComplex * input3,
                 vDSP_Stride stride3,
                 DSPDoubleSplitComplex * result,
                 vDSP_Stride strideResult,
                 vDSP_Length size);
```

Discussion

This performs the operation

$$D_{nL} = A_{nI}^* B_{nJ} + C_{nK} \quad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvcmul

Complex vector conjugate and multiply; single precision.

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

Parameters

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

Discussion

Multiplies vector B by the complex conjugates of vector A and stores the results in vector B.

$$C_{nK} = A_{nI}^* B_{nJ}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvcmulD

Complex vector conjugate and multiply; double precision.

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

Parameters

<i>A</i>	Double-precision complex input vector
----------	---------------------------------------

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

Discussion

Multiplies vector *B* by the complex conjugates of vector *A* and stores the results in vector *B*.

$$C_{nK} = A_{nI}^* B_{nJ}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvmul

Multiplies complex vectors *A* and *B* and leaves the result in complex vector *C*; single precision.

```
void vDSP_zvmul (DSPSplitComplex * A,
                vDSP_Stride I,
                DSPSplitComplex * B,
                vDSP_Stride J,
                DSPSplitComplex * C,
                vDSP_Stride K,
                vDSP_Length N,
                int conjugate);
```

Discussion

Pass 1 or -1 for *F*, for normal or conjugate multiplication, respectively. Results are undefined for other values of *F*.

$$Re[C_{nK}] = Re[A_{nI}] Re[B_{nJ}] - F(Im[A_{nI}] Im[B_{nJ}])$$

$$Im[C_{nK}] = Re[A_{nI}] Im[B_{nJ}] + F(Im[A_{nI}] Re[B_{nJ}])$$

n = {0, N-1}

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvmulD

Multiplies complex vectors A and B and leaves the result in complex vector C; double precision.

```
void vDSP_zvmulD (DSPDoubleSplitComplex * input1,
                 vDSP_Stride stride1,
                 DSPDoubleSplitComplex * input2,
                 vDSP_Stride stride2,
                 DSPDoubleSplitComplex * result,
                 vDSP_Stride strideResult,
                 vDSP_Length size,
                 int conjugate);
```

Discussion

Pass 1 or -1 for F, for normal or conjugate multiplication, respectively. Results are undefined for other values of F.

$$Re[C_{nK}] = Re[A_{nI}]Re[B_{nJ}] - F(Im[A_{nI}]Im[B_{nJ}])$$

$$Im[C_{nK}] = Re[A_{nI}]Im[B_{nJ}] + F(Im[A_{nI}]Re[B_{nJ}])$$

$$n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvsub

Subtracts complex vector B from complex vector A and leaves the result in complex vector C; single precision.

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

Parameters

A

Input vector

I

Address stride for A

B

Input vector

J

Address stride for B

C

Output vector

K
Address stride for C

N
Complex element count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvsubD

Subtracts complex vector B from complex vector A and leaves the result in complex vector C; double precision.

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

Parameters

A
Input vector

I
Address stride for A

B
Input vector

J
Address stride for B

C
Output vector

K
Address stride for C

N
Complex element count

Discussion

This performs the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

Document Revision History

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

Date	Notes
2009-01-06	Added more detailed descriptions of function parameters.
2007-06-15	This document describes the V API for the vDSP functions that perform arithmetic operations combining the elements of two vectors

REVISION HISTORY

Document Revision History

Index

V

- vDSP_deq22 function 12
- vDSP_deq22D function 13
- vDSP_vaam function 14
- vDSP_vaamD function 15
- vDSP_vadd function 16
- vDSP_vaddD function 17
- vDSP_vam function 17
- vDSP_vamD function 18
- vDSP_vasbm function 18
- vDSP_vasbmD function 19
- vDSP_vasm function 21
- vDSP_vasmD function 21
- vDSP_vdist function 22
- vDSP_vdistD function 23
- vDSP_vdiv function 24
- vDSP_vdivD function 25
- vDSP_vdivi function 26
- vDSP_venvlp function 27
- vDSP_venvlpD function 29
- vDSP_veqvi function 30
- vDSP_vintb function 30
- vDSP_vintbD function 31
- vDSP_vma function 32
- vDSP_vmaD function 33
- vDSP_vmax function 34
- vDSP_vmaxD function 35
- vDSP_vmaxmg function 36
- vDSP_vmaxmgD function 37
- vDSP_vmin function 38
- vDSP_vminD function 39
- vDSP_vminmg function 40
- vDSP_vminmgD function 41
- vDSP_vmma function 42
- vDSP_vmmaD function 44
- vDSP_vmmsb function 45
- vDSP_vmmsbD function 46
- vDSP_vmsa function 47
- vDSP_vmsaD function 48
- vDSP_vmsb function 49
- vDSP_vmsbD function 50
- vDSP_vmul function 51
- vDSP_vmulD function 51
- vDSP_vpoly function 52
- vDSP_vpolyD function 53
- vDSP_vpythg function 54
- vDSP_vpythgD function 55
- vDSP_vqint function 57
- vDSP_vqintD function 58
- vDSP_vsbm function 59
- vDSP_vsbmD function 60
- vDSP_vsbmD function 61
- vDSP_vsbmD function 62
- vDSP_vsbmD function 63
- vDSP_vsbmD function 64
- vDSP_vsub function 65
- vDSP_vsubD function 65
- vDSP_vswap function 65
- vDSP_vswapD function 66
- vDSP_vtmerg function 67
- vDSP_vtmergD function 68
- vDSP_zaspec function 69
- vDSP_zaspecD function 69
- vDSP_zcoher function 70
- vDSP_zcoherD function 70
- vDSP_zcspec function 71
- vDSP_zcspecD function 72
- vDSP_zrvadd function 72
- vDSP_zrvaddD function 73
- vDSP_zrvdiv function 74
- vDSP_zrvdivD function 74
- vDSP_zrvmul function 75
- vDSP_zrvmulD function 75
- vDSP_zrvsub function 76
- vDSP_zrvsubD function 77
- vDSP_ztrans function 78
- vDSP_ztransD function 78
- vDSP_zvadd function 79
- vDSP_zvaddD function 80
- vDSP_zvcma function 80
- vDSP_zvcmaD function 81
- vDSP_zvcmul function 81

vDSP_zvcmulD function 82
vDSP_zvmul function 83
vDSP_zvmulD function 84
vDSP_zvsub function 84
vDSP_zvsubD function 85