vDSP Reference Collection

Performance > Vector Engines

2008-10-15

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

Contents

Introduction

Framework:	Accelerate/vecLib
Declared in	vDSP.h

This document describes the C API for the digital signal processing functionality in vecLib.

Other References

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 21)
  Vector absolute values; single precision.

`vDSP_vabsD` (page 22)
  Vector absolute values; double precision.

`vDSP_vabsi` (page 22)
  Integer vector absolute values.

`vDSP_zvabs` (page 77)
  Complex vector absolute values; single precision.

`vDSP_zvabsD` (page 77)
  Complex vector absolute values; double precision.

`vDSP_vnabs` (page 53)
  Vector negative absolute values; single precision.

`vDSP_vnabsD` (page 53)
  Vector negative absolute values; double precision.

Negating Values of Elements

`vDSP_vneg` (page 54)
  Vector negative values; single precision.

`vDSP_vnegD` (page 55)
  Vector negative values; double precision.

Filling or Clearing Elements

Building a Vector With Generated Values

Squaring Element Values

Converting Between Polar and Rectangular Coordinates

Converting to Decibel Equivalents

Extracting Fractional Parts of Elements

Conjugating Complex Vectors

`vDSP_zvconjD` (page 79)
  Complex vector conjugate; double precision.

Squaring Magnitudes of Complex Elements

`vDSP_zvmags` (page 81)
  Complex vector magnitudes squared; single precision.

`vDSP_zvmagsD` (page 81)
  Complex vector magnitudes squared; double precision.

`vDSP_zvmgsa` (page 82)
  Complex vector magnitudes square and add; single precision.

`vDSP_zvmgsaD` (page 83)
  Complex vector magnitudes square and add; double precision.

Calculating Phase Values of Complex Elements

`vDSP_zvphas` (page 87)
  Complex vector phase; single precision.

`vDSP_zvphasD` (page 87)
  Complex vector phase; double precision.

Clipping, Limit, and Threshold Operations

`vDSP_vclip` (page 25)
  Vector clip; single precision.

`vDSP_vclipD` (page 28)
  Vector clip; double precision.

`vDSP_vclipc` (page 26)
  Vector clip and count; single precision.

`vDSP_vclipcD` (page 27)
  Vector clip and count; double precision.

`vDSP_viclip` (page 45)
  Vector inverted clip; single precision.

`vDSP_viclipD` (page 46)
  Vector inverted clip; double precision.

`vDSP_vlim` (page 49)
  Vector test limit; single precision.

`vDSP_vlimD` (page 49)
  Vector test limit; double precision.

`vDSP_vthr` (page 70)
  Vector threshold; single precision.

`vDSP_vthrD` (page 71)
  Vector threshold; double precision.

Compressing Element Values

Gathering Elements of a Vector

Using Indices to Select Elements of a Vector

Reversing the Order of Elements

Copying Complex Vectors

Finding Zero Crossings

Calculating Linear Average of a Vector

Performing Linear Interpolation Between Neighboring Elements

Integrating a Vector

Sorting a Vector

Calculating Sliding-Window Sums

Converting Between Single and 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`

B
: Maximum number of crossings to find

C
: Index of last crossing found

D
: Total number of zero crossings found

N
: Count of elements in A

Discussion
This performs the operation

$d = c = 0$

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

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

$d = d+1$

If $d = b$ then

$c = nI$,

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

The "function" 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

A
: Double-precision real input vector

I
: Stride for A

B
: Maximum number of crossings to find

C
: Index of last crossing found

D
: Total number of zero crossings found

N
: Count of elements in A

Discussion

This performs the operation

$d = c = 0$

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

If $\mathrm{sign}(A_{nI})! = \mathrm{sign}(A_{(n-1)I})$ then

$d = d+1$

If $d = b$ then

$c = nI,$

exit. n = {1, N-1}

The "function" 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

A

Single-precision real input vector

I

Stride for A, must be even

C

Single-precision output vector

K

Stride for C, must be even

N

Number of ordered pairs processed

Discussion

This performs the operation

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

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

A

Double-precision real input vector

I

Stride for `A`, must be even

C

Double-precision output vector

K

Stride for `C`, must be even

N

Number of ordered pairs processed

Discussion

This performs the operation

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

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

A

Single-precision real input vector

I

Stride for A, must be even

C

Single-precision real output vector

K

Stride for C, must be even

N

Number of ordered pairs processed

Discussion

This performs the operation

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

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

A

Double-precision real input vector

I

Stride for A, must be even

C

Double-precision real output vector

K

Stride for C, must be even

N

Number of ordered pairs processed

Discussion
This performs the operation

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

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

A
Single-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

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

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

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

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

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

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

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

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

A
Double-precision real input vector

I
Stride for `A`

B
Double-precision real input scalar

C
Double-precision real input-output vector

K
Stride for `C`

N
Count ; each vector must have at least N elements

Discussion
This performs the operation

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

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

$$\begin{cases} \text{If} \quad A_{nI} < b & \text{then} \quad D_{nM} = B \\ \text{If} \quad A_{nI} > c & \text{then} \quad D_{nM} = C \; , \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\} \\ \text{If} \quad b \le A_{nI} \le c & \text{then} \quad D_{nM} = A_{nI} \end{cases}$$

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

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 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

$$\begin{cases} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C\ , \qquad n = \{0, \text{N-1}\} \\ \text{If } b \le A_{nI} \le c & \text{then } D_{nM} = A_{nI} \end{cases}$$

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

$$\begin{cases} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C \ , \qquad n = \{0, N\text{-}1\} \\ \text{If } b \le A_{nI} \le c & \text{then } D_{nM} = A_{nI} \end{cases}$$

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

$$\begin{cases} \text{If} \quad A_{nI} < b & \text{then} \quad D_{nM} = B \\ \text{If} \quad A_{nI} > c & \text{then} \quad D_{nM} = C \;, \qquad n = \{0, N\text{-}1\} \\ \text{If} \quad b \le A_{nI} \le c & \text{then} \quad D_{nM} = A_{nI} \end{cases}$$

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

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

$$p = 0$$

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

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

A
: Double-precision real 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

Performs the operation

$$p = 0$$

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

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

A
Single-precision real input vector

I
Stride for A

B
Single-precision real input scalar: zero reference

C
Single-precision real output vector

K
Stride for C

N
Count

F
Power (0) or amplitude (1) flag

Discussion

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

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

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

A

Double-precision real input vector

I

Stride for A

B

Double-precision real input scalar: zero reference

C

Double-precision real output vector

K

Stride for C

N

Count

F

Power (0) or amplitude (1) flag

Discussion

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

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

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

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

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

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

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

A
Double-precision real input scalar

C
Double-precision real output vector

K
Stride for `C`

N
Count

Discussion
Performs the operation

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

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

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

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

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

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

A
Double-precision real input vector

I
Stride for `A`

C
Double-precision real output vector

K
Stride for `C`

N
Count

Discussion
Performs the operation

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

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_vgathr

Vector gather; single precision.

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

Parameters

A
: Single-precision real input vector

B
: Integer vector containing indices

J
: Stride for `B`

C
: Single-precision real output vector

K
: Stride for `C`

N
: Count

Discussion
Performs the operation

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

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_vgathra

Vector gather, absolute pointers; single precision.

```
void  vDSP_vgathra (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

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

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_vgathraD

Vector gather, absolute pointers; double precision.

```
void  vDSP_vgathraD (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

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

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_vgathrD

Vector gather; double precision.

```
void  vDSP_vgathrD (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

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

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

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

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

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

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

$$C_{nK} = A_0 \qquad \text{for } 0 \le n \le \operatorname{trunc}(B_0)$$

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

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

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

$$B_\Delta = B_{[m+1]J} - B_{mI} \qquad m = \{0, M\text{-}2\}$$

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 [-infinity, trunc(B[0*J]], the function is the constant A[0*I].
- In each interval (trunc(B[m*J]), 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 <= m < M-1.)
- In the interval (B[(M-1)*J], infinity], the function is the constant A[(M-1)*I].
- For 0 <= 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

A
: Double-precision real 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 for `C`

M
: Count for `A` and `B`

Discussion

Performs the operation

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

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

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

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

$$B_\Delta = B_{[m+1]J} - B_{mI} \qquad \text{m} = \{0, \text{M-2}\}$$

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 [-infinity, trunc(B[0*J]], the function is the constant A[0*I].
- In each interval (trunc(B[m*J]), 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 <= m < M-1.)
- In the interval (B[(M-1)*J], infinity], the function is the constant A[(M-1)*I].
- For 0 <= 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

A
: Single-precision real input vector

I
: Stride for A

B
: Single-precision real input scalar: lower threshold

C
: Single-precision real input scalar: upper threshold

D
: Single-precision real output vector

L
: Stride for D

N
: Count

Discussion

Performs the operation

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

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

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

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

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

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

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

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

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

A
: Double-precision real input vector

B
: Double-precision real input vector: indices

J
: Stride for B

C
: Double-precision real output vector

K
: Stride for C

N
: Count

Discussion
Performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for `A`

B
: Single-precision real input scalar: limit

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for `D`

N
: 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

A
: Double-precision real input vector

I
: Stride for A

B
: Double-precision real input scalar: limit

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for D

N
: 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

A
: Single-precision real input vector

B
: Single-precision real input vector: integer parts are indices into `A` and fractional parts are interpolation constants

J
: Stride for `B`

C
: Single-precision real output vector

K
: Stride for `C`

N
: Count for C

M
: Length of A

Discussion

Performs the operation

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

$$\text{where: } \beta = \mathrm{trunc}(B_{nJ})$$
$$\alpha = B_{nJ} - \mathrm{float}(\beta)$$

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

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

Discussion

Performs the operation

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

$$\text{where: } \beta = \operatorname{trunc}(B_{nJ})$$
$$\alpha = B_{nJ} - \operatorname{float}(\beta)$$

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

A

Single-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

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

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

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

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

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

A
    Double-precision real input vector

I
    Stride for A

C
    Double-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_vramp

Build ramped vector; single precision.

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

Parameters

A
    Single-precision real input scalar: initial value

B
    Single-precision real input scalar: increment or decrement

C
    Single-precision real output vector

K
    Stride for C

N

Count

Discussion

Performs the operation

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

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

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

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

$$C_0 = 0$$

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

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

A

Double-precision real input vector

I

Stride for `A`

S

Double-precision real input scalar: weighting factor

C

Double-precision real output vector

K

Stride for `C`

N

Count

Discussion

Performs the operation

$$C_0 = 0$$

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

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

C

Single-precision real input-output vector

K

Stride for `C`

N

Count

Discussion

Performs the operation

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

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

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

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

A
: Single-precision real input vector

I
: Stride for A

B
: Single-precision real input scalar

C
: Single-precision real output vector

K
: Stride for C

N
: Count

Discussion

Performs the operation

$$C_0 = 0.0$$

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

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

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

A
: Double-precision real input vector

I
: Stride for `A`

B
: Double-precision real input scalar

C
: Double-precision real output vector

K
: Stride for `C`

N
: Count

Discussion

Performs the operation

$$C_0 = 0.0$$

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

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

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

C
: Single-precision real input-output vector

N
: Count

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

Discussion
Performs 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

Discussion
Performs 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

Discussion

Leaves 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

Discussion

Leaves 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

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

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_vsq

Computes 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

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

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

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

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

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

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_vssqD

Computes the signed squares of vector `signal1` 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

$$C_{nK} = A_{nI} \cdot \left|A_{nI}\right| \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

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

A
: Single-precision real input vector

I
: Stride for `A`

C
: Single-precision real output vector

K
: Stride for `C`

N
: Count of output points

P
: Length of window

Discussion
Performs the operation

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

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

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

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

A
: Single-precision real input vector

I
: Stride for A

S1
: Single-precision real input scalar: scale factor

S2
: Single-precision real input scalar: base offset

C
: Single-precision real input vector: lookup table

M
: Lookup table size

D
: Single-precision real output vector

L
: Stride for D

N
: Count

Discussion

Performs the operation

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

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_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

A
: Double-precision real input vector

I
: Stride for A

S1
: Double-precision real input scalar: scale factor

S2
: Double-precision real input scalar: base offset

C
: Double-precision real input vector: lookup table

M
: Lookup table size

D
: Double-precision real output vector

L
: Stride for D

N
: Count

Discussion
Performs the operation

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

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

A
Single-precision real input vector

I
Stride for A

B
Single-precision real input scalar: lower threshold

C
Single-precision real output vector

K
Stride for C

N
Count

Discussion
Performs the operation

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

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_vthrD

Vector threshold; double precision.

```
void  vDSP_vthrD (double * A,
vDSP_Stride I,
double * B,
double * C,
vDSP_Stride K,
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 output vector

K
Stride for C

N
Count

Discussion
Performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for A

B
: Single-precision real input scalar: lower threshold

C
: Single-precision real output vector

K
: Stride for C

N
: Count

Discussion

Performs the operation

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

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

A
: Double-precision real input vector

I
: Stride for A

B
: Double-precision real input scalar: lower threshold

C
: Double-precision real output vector

K
: Stride for C

N
: Count

Discussion

Performs the operation

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

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

A
: Single-precision real input vector

I

Stride for `A`

B

Single-precision real input scalar: lower threshold

C

Single-precision real input scalar

D

Single-precision real output vector

L

Stride for `D`

N

Count

Discussion

Performs the operation

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

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

A

Double-precision real input vector

I

Stride for `A`

B

Double-precision real input scalar: lower threshold

C

Double-precision real input scalar

D

Double-precision real output vector

L

Stride for `D`

N

Count

Discussion

Performs the operation

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

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

$$C_0 = 0.0$$

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

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

A
Double-precision real input vector

I
Stride for A

B
Double-precision real input scalar: step size

C
Double-precision real output vector

K
Stride for C

N
Count

Discussion
Performs the operation

$$C_0 = 0.0$$

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

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

A
: Single-precision complex input vector

I
: Stride for `A`

C
: Single-precision real output vector

K
: Stride for C

N
: Count

Discussion
This peforms the operation

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

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 peforms the operation

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

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.

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

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.

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

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

A
Single-precision complex input vector

I
Stride for A

C
Single-precision real output vector

K
Stride for C

N
Count

Discussion
Calculates the squared magnitudes of complex vector A.

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

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`.

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

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.

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

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.

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

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zvmov

Complex vector copy; single precision.

```
void  vDSP_zvmov (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
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_zvmovD

Complex vector copy; double precision.

```
void  vDSP_zvmovD (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

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

A
Single-precision complex input vector

I
Stride for A

C
Single-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_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

A
: Single-precision complex input vector

I
: Stride for `A`

C
: Single-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.

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

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

A
: Double-precision complex input vector

I
: Stride for `A`

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.

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP Vector Scalar Arithmetic Operations Reference

Framework: Accelerate/vecLib

Declared in vDSP.h

Overview

Describes the C API for the vecLib functions that perform arithmetic operations combining a scalar with each element of a vector.

Functions by Task

Adding a Scalar to Elements of a Vector

`vDSP_vsadd` (page 92)
: Vector scalar add; single precision.

`vDSP_vsaddD` (page 93)
: Vector scalar add; double precision.

`vDSP_vsaddi` (page 94)
: Integer vector scalar add.

Dividing Elements of a Vector by a Scalar

`vDSP_vsdiv` (page 94)
: Vector scalar divide; single precision.

`vDSP_vsdivD` (page 95)
: Vector scalar divide; double precision.

`vDSP_vsdivi` (page 96)
: Integer vector scalar divide.

`vDSP_zvdiv` (page 103)
: Complex vector divide; single precision.

`vDSP_zvdivD` (page 104)
: Complex vector divide; double precision.

`vDSP_svdiv` (page 90)
: Divide scalar by vector; single precision.

Multiplying Elements of a Vector by a Scalar

Multiplying Elements of a Vector by a Scalar, then Adding or Subtracting Another Scalar

Functions

vDSP_svdiv

Divide scalar by vector; single precision.

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

Parameters

A
: Single-precision real input scalar

B
: Single-precision real input vector

J
: Stride for `B`

C
: Single-precision real output vector

K
: Stride for `C`

N
: Count

Discussion

This performs the operation

$$C_{nK} = \frac{A}{B_{nJ}} , \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

Divides scalar `A` by each element of vector `B`, storing the results in vector `C`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_svdivD

Divide scalar by vector; double precision.

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

Parameters

A
: Double-precision real input scalar

B
: Double-precision real input vector

J
: Stride for B

C
: Double-precision real output vector

K
: Stride for C

N
: Count

Discussion

This performs the operation

$$C_{nK} = \frac{A}{B_{nJ}}, \qquad n = \{0, N\text{-}1\}$$

Divides scalar A by each element of vector B, storing the results in vector C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsadd

Vector scalar add; single precision.

```
void vDSP_vsadd (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 output vector

K
: Stride for C

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} + B \qquad n = \{0, N\text{-}1\}$$

Adds scalar B to each element of vector A 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_vsaddD

Vector scalar add; double precision.

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

Parameters

A
Double-precision real input vector

I
Stride for A

B
Double-precision real input scalar

C
Double-precision real output vector

K
Stride for C

N
Count

Discussion
Performs the operation

$$C_{nK} = A_{nI} + B \qquad n = \{0, N\text{-}1\}$$

Adds scalar B to each element of vector A 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_vsaddi

Integer vector scalar add.

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

Parameters

A
Integer input vector

I
Stride for A

B
Integer input scalar

C
Integer output vector

K
Stride for C

N
Count

Discussion

Performs the operation

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

Adds scalar B to each element of vector A 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_vsdiv

Vector scalar divide; single precision.

```
void vDSP_vsdiv (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 output vector

K
Stride for C

N
Count

Discussion
Performs the operation

$$C_{nK} = \frac{A_{nI}}{B} \qquad n = \{0, N\text{-}1\}$$

Divides each element of vector A by scalar 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_vsdivD

Vector scalar divide; double precision.

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

Parameters

A
Double-precision real input vector

I
Stride for A

B
Double-precision real input scalar

C
Double-precision real output vector

K
Stride for C

N
Count

Discussion
Performs the operation

$$C_{nK} = \frac{A_{nI}}{B} \qquad n = \{0, N\text{-}1\}$$

Divides each element of vector `A` by scalar `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_vsdivi

Integer vector scalar divide.

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

Parameters

A
: Integer input vector

I
: Stride for `A`

B
: Integer input scalar

C
: Integer output vector

K
: Stride for `C`

N
: Count

Discussion
Performs the operation

$$C_{nK} = \frac{A_{nI}}{B} \qquad n = \{0, N\text{-}1\}$$

Divides each element of vector `A` by scalar `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_vsma

Vector scalar multiply and vector add; single precision.

```
void vDSP_vsma (const float * A,
vDSP_Stride I,
const float * B,
const float * C,
vDSP_Stride K,
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

C
: Single-precision real input vector

K
: Stride for `C`

D
: Single-precision real output vector

L
: Stride for `D`

N
: Count

Discussion
Performs the operation

$$D_{nM} = A_{nI} \cdot B + C_{nK} \qquad \mathrm{n} = \{0, \mathrm{N}-1\}$$

Multiplies vector `A` by scalar `B` and then adds the products to vector `C`. Results are stored in vector `D`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_vsmaD

Vector scalar multiply and vector add; double precision.

```
void vDSP_vsmaD (const double * A,
vDSP_Stride I,
const double * B,
const double * C,
vDSP_Stride K,
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

C
    Double-precision real input vector

K
    Stride for C

D
    Double-precision real output vector

L
    Stride for D

N
    Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \cdot B + C_{nK} \qquad n = \{0, N\text{-}1\}$$

Multiplies vector A by scalar B and then adds the products to vector C. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsmsa

Vector scalar multiply and scalar add; single precision.

```
void vDSP_vsmsa (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

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for D

N
: Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \cdot B + C \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

Multiplies vector A by scalar B and then adds scalar C to each product. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsmsaD

Vector scalar multiply and scalar add; double precision.

```
void vDSP_vsmsaD (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

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for D

N
: Count

Discussion
Performs the operation

$$D_{nM} = A_{nI} \cdot B + C \qquad n = \{0, N\text{-}1\}$$

Multiplies vector A by scalar B and then adds scalar C to each product. Results are stored in vector D.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_vsmsb

Vector scalar multiply and vector subtract; single precision.

```
void vDSP_vsmsb (float * A,
vDSP_Stride I,
float * B,
float * C,
vDSP_Stride K,
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

C
Single-precision real input vector

K
Stride for C

D
Single-precision real output vector

L
Stride for D

N
Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \cdot B - C_{nK} \qquad n = \{0, N\text{-}1\}$$

Multiplies vector A by scalar B and then subtracts vector C from the products. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_vsmsbD

Vector scalar multiply and vector subtract; double precision.

```
void vDSP_vsmsbD (double * A,
vDSP_Stride I,
double * B,
double * C,
vDSP_Stride K,
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

C
: Double-precision real input vector

K
: Stride for C

D
: Double-precision real output vector

L
: Stride for D

N
: Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \bullet B - C_{nK} \qquad n = \{0, N\text{-}1\}$$

Multiplies vector A by scalar B and then subtracts vector C from the products. Results are stored in vector D.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_vsmul

Multiplies vector `signal1` by scalar `signal2` and leaves the result in vector `result`; single precision.

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

Discussion
This performs the operation

$$C_{nK} = A_{nI} \cdot B \qquad n = \{0, N\text{-}1\}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_vsmulD

Multiplies vector `signal1` by scalar signal2 and leaves the result in vector `result`; double precision.

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

Discussion
This performs the operation

$$C_{nK} = A_{nI} \cdot B \qquad n = \{0, N\text{-}1\}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zvdiv

Complex vector divide; single precision.

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

Parameters

A
: Single-precision complex input vector

I
: Stride for A

B
: Single-precision complex input vector

J
: Stride for B

C
: Single-precision complex output vector

K
: Stride for C

N
: Count

Discussion

Divides vector B by vector A.

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_zvdivD

Complex vector divide; double precision.

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

Parameters

A
Double-precision complex input vector

I
Stride for A

B
Double-precision complex input vector

J
Stride for B

C
Double-precision complex output vector

K
Stride for C

N
Count

Discussion
Divides vector B by vector A.

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

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_zvsma

Complex vector scalar multiply and add; single precision.

```
void vDSP_zvsma (DSPSplitComplex * A,
vDSP_Stride I,
DSPSplitComplex * B,
DSPSplitComplex * C,
vDSP_Stride K,
DSPSplitComplex * D,
vDSP_Stride L,
vDSP_Length N);
```

Parameters

A
: Single-precision complex input vector

I
: Stride for A

B
: Single-precision complex input scalar

C
: Single-precision real input vector

K
: Stride for C

D
: Single-precision real output vector

L
: Stride for C

N
: Count

Discussion

Multiplies vector A by scalar B and add the products to vector C. The result is stored in vector D.

$$D_{nL} = A_{nI}B + C_{nK}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvsmaD

Complex vector scalar multiply and add; double precision.

```
void vDSP_zvsmaD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
DSPDoubleSplitComplex * B,
DSPDoubleSplitComplex * C,
vDSP_Stride K,
DSPDoubleSplitComplex * D,
vDSP_Stride L,
vDSP_Length N);
```

Parameters

A
Double-precision complex input vector

I
Stride for A

B
Double-precision complex input scalar

C
Double-precision real input vector

K
Stride for C

D
Double-precision real output vector

L
Stride for C

N
Count

Discussion

Multiplies vector A by scalar B and add the products to vector C. The result is stored in vector D.

$$D_{nL} = A_{nI}B + C_{nK}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_zvzsml

Complex vector multiply by complex scalar; single precision.

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

Parameters

A

Single-precision complex input vector

I

Stride for A

B

Single-precision complex input scalar

C

Single-precision complex output vector

K

Stride for C

N

Count

Discussion

This peforms the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zvzsmlD

Complex vector multiply by complex scalar; double precision.

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

Parameters

A

Double-precision complex input vector

I

Stride for A

B

Double-precision complex input scalar

`C`
: Double-precision complex output vector

`K`
: Stride for `C`

`N`
: Count

Discussion

This peforms the operation

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP Vector-To-Scalar 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 compute scalars as output. Examples of such operations include calculating the dot product of a vector, or finding the minimum or maximum of a vector.

Functions by Task

Calculating Dot Products

`vDSP_dotpr` (page 114)
: Computes the dot or scalar product of vectors `A` and `B` and leaves the result in scalar `C`; single precision.

`vDSP_dotprD` (page 114)
: Computes the dot or scalar product of vectors `A` and `B` and leaves the result in scalar `C`; double precision.

`vDSP_zdotpr` (page 139)
: Calculates the complex dot product of complex vectors signal1 and signal2 and leaves the result in complex vector `result`; single precision.

`vDSP_zdotprD` (page 139)
: Calculates the complex dot product of complex vectors signal1 and signal2 and leaves the result in complex vector `result`; double precision.

`vDSP_zidotpr` (page 140)
: Calculates the conjugate dot product (or inner dot product) of complex vectors `signal1` and `signal2` and leave the result in complex vector `result`; single precision.

`vDSP_zidotprD` (page 140)
: Calculates the conjugate dot product (or inner dot product) of complex vectors signal1 and signal2 and leave the result in complex vector `result`; double precision.

`vDSP_zrdotpr` (page 141)
: Calculates the complex dot product of complex vector `A` and real vector `B` and leaves the result in complex vector `C`; single precision.

`vDSP_zrdotprD` (page 141)
: Calculates the complex dot product of complex vector `A` and real vector `B` and leaves the result in complex vector `C`; double precision.

Finding Maximums

Finding Minimums

Calculating Means

Summing Vectors

Functions

vDSP_dotpr

Computes the dot or scalar product of vectors A and B and leaves the result in scalar C; single precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \cdot B_{nJ} \qquad n = \{0, N-1\}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_dotprD

Computes the dot or scalar product of vectors A and B and leaves the result in scalar C; double precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \cdot B_{nJ} \qquad n = \{0, N-1\}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_maxmgv

Vector maximum magnitude; single precision.

```
void vDSP_maxmgv (const float * A,
vDSP_Stride I,
float * C,
vDSP_Length N);
```

Parameters

A
: Single-precision real input vector. If passed a vector with no elements, this function returns a value of 0 in `C`.

I
: Stride for `A`

C
: Output scalar

N
: Count

Discussion

This performs the operation

```
*C = 0;
for (n = 0; n < N; ++n)
    if (*C < abs(A[n*I]))
        *C = abs(A[n*I]);
```

Finds the element with the greatest magnitude in vector `A` and copies this value to scalar `C`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_maxmgvD

Vector maximum magnitude; double precision.

```
void vDSP_maxmgvD (const double * A,
vDSP_Stride I,
double * C,
vDSP_Length N);
```

Parameters

A
: Double-precision real input vector. If passed a vector with no elements, this function returns a value of 0 in `C`.

I
: Stride for `A`

C
: Output scalar

N

Count

Discussion

This performs the operation

```
*C = 0;
for (n = 0; n < N; ++n)
    if (*C < abs(A[n*I]))
        *C = abs(A[n*I]);
```

Finds the element with the greatest magnitude in vector `A` and copies this value to scalar `C`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_maxmgvi

Vector maximum magnitude with index; single precision.

```
void vDSP_maxmgvi (float * A,
vDSP_Stride I,
float * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A

Single-precision real input vector. If passed a vector with no elements, this function returns a value of 0 in `*C`, and the value returned in `IC` is undefined.

I

Stride for `A`

C

Output scalar

IC

Output scalar index

N

Count

Discussion

This performs the operation

```
*C = 0;
for (n = 0; n < N; ++n)
{
    if (*C < abs(A[n*I]))
    {
        *C = abs(A[n*I]);
        *IC = n*I;
    }
}
```

Copies the element with the greatest magnitude from real vector `A` to real scalar `C`, and writes its zero-based index to integer scalar `IC`. The index is the actual array index, not the pre-stride index. If vector `A` contains more than one instance of the maximum magnitude, `IC` contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_maxmgviD

Vector maximum magnitude with index; double precision.

```
void vDSP_maxmgviD (double * A,
vDSP_Stride I,
double * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A
: Double-precision real input vector. If passed a vector with no elements, this function returns a value of 0 in `*C`. The value returned in `IC` is undefined.

I
: Stride for `A`

C
: Output scalar

IC
: Output scalar

N
: Count

Discussion
This performs the operation

```
*C = 0;
for (n = 0; n < N; ++n)
{
    if (*C < abs(A[n*I]))
    {
        *C = abs(A[n*I]);
        *IC = n*I;
    }
}
```

Copies the element with the greatest magnitude from real vector `A` to real scalar `C`, and writes its zero-based index to integer scalar `IC`. The index is the actual array index, not the pre-stride index. If vector `A` contains more than one instance of the maximum magnitude, `IC` contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_maxv

Vector maximum value; single precision.

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

Parameters

A
: Single-precision real input vector.

I
: Stride for `A`

C
: Output scalar

N
: Count

Discussion
This performs the operation

```
*C = -INFINITY;
for (n = 0; n < N; ++n)
    if (*C < A[n*I])
        *C = A[n*I];
```

Finds the element with the greatest value in vector `A` and copies this value to scalar `C`. If `A` has length of 0, this function returns `-INFINITY`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_maxvD

Vector maximum value; double precision.

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

Parameters

A
: Double-precision real input vector

I
    Stride for A

C
    Output scalar

N
    Count

Discussion

This performs the operation

```
*C = -INFINITY;
for (n = 0; n < N; ++n)
    if (*C < A[n*I])
        *C = A[n*I];
```

Finds the element with the greatest value in vector A and copies this value to scalar C. If A has length of 0, this function returns -INFINITY.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_maxvi

Vector maximum value with index; single precision.

```
void vDSP_maxvi (float * A,
vDSP_Stride I,
float * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A
    Single-precision real input vector.

I
    Stride for A

C
    Output scalar

IC
    Output scalar index

N
    Count

Discussion

This performs the operation

```
*C = -INFINITY;
for (n = 0; n < N; ++n)
{
    if (*C < A[n * I])
```

```
    {
        *C = A[n * I];
        *IC = n * I;
    }
}
```

Copies the element with the greatest value from real vector `A` to real scalar `C`, and writes its zero-based index to integer scalar `IC`. The index is the actual array index, not the pre-stride index. If vector `A` contains more than one instance of the maximum value, `IC` contains the index of the first instance. If `A` is vector with no elements, this function returns a value of -infinity in `C` and `*IC` is undetermined.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_maxviD

Vector maximum value with index; double precision.

```
void vDSP_maxviD (double * A,
vDSP_Stride I,
double * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for `A`

C
: Output scalar

IC
: Output scalar

N
: Count

Discussion
This performs the operation

```
*C = -INFINITY;
for (n = 0; n < N; ++n)
{
    if (*C < A[n * I])
    {
        *C = A[n * I];
        *IC = n * I;
    }
}
```

Copies the element with the greatest value from real vector A to real scalar C, and writes its zero-based index to integer scalar IC. The index is the actual array index, not the pre-stride index. If vector A contains more than one instance of the maximum value, IC contains the index of the first instance. If A is vector with no elements, this function returns a value of -infinity in C and *IC is undetermined.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_meamgv

Vector mean magnitude; single precision.

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

Parameters

A
Single-precision real input vector

I
Stride for A

C
Output scalar

N
Count

Discussion
This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} |A_{ni}|$$

Finds the mean of the magnitudes of elements of vector A and stores this value in scalar C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_meamgvD

Vector mean magnitude; double precision.

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

Parameters

A
: Double-precision real input vector

I
: Stride for `A`

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} \left| A_{ni} \right|$$

Finds the mean of the magnitudes of elements of vector `A` and stores this value in scalar `C`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_meanv

Vector mean value; single precision.

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

Parameters

A
: Single-precision real input vector. If passed an array of length 0, this function returns a NaN.

I
: Stride for `A`

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} A_{ni}$$

Finds the mean value of the elements of vector A and stores this value in scalar C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_meanvD

Vector mean value; double precision.

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

Parameters

A
Double-precision real input vector

I
Stride for A

C
Output scalar

N
Count

Discussion
This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} A_{ni}$$

Finds the mean value of the elements of vector A and stores this value in scalar C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_measqv

Vector mean square value; single precision.

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

Parameters

A

Single-precision real input vector. If passed an array of length 0, this function returns a NaN.

I

Stride for `A`

C

Output scalar

N

Count

Discussion

This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} A_{ni}^2$$

Finds the mean value of the squares of the elements of vector `A` and stores this value in scalar `C`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_measqvD

Vector mean square value; double precision.

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

Parameters

A

Double-precision real input vector

I

Stride for `A`

C

Output scalar

N

Count

Discussion

This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} A_{ni}^2$$

Finds the mean value of the squares of the elements of vector `A` and stores this value in scalar `C`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_minmgv

Vector minimum magnitude; single precision.

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

Parameters

A
: Single-precision real input vector. If passed an array of length 0, this function returns +INF.

I
: Stride for `A`

C
: Output scalar

N
: Count

Discussion
This performs the operation

$$c = |A_0| \quad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \qquad \text{n} = \{1, \text{N-1}\}$$

Finds the element with the least magnitude in vector `A` and copies this value to scalar `C`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_minmgvD

Vector minimum magnitude; double precision.

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

Parameters

A
: Double-precision real input vector

I
: Stride for A

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$c = |A_0| \quad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \qquad n = \{1, N\text{-}1\}$$

Finds the element with the least magnitude in vector A and copies this value to scalar C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_minmgvi

Vector minimum magnitude with index; single precision.

```
void vDSP_minmgvi (float * A,
vDSP_Stride I,
float * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A
: Single-precision real input vector. If passed a zero vector, this function returns a value of +INF with an indeterminate index.

I
: Stride for A

C
: Output scalar

IC
: Output scalar index

N
: Count

Discussion
This performs the operation

$$c = |A_0| \quad d = 0 \qquad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \quad d = ni \qquad n = \{1, N-1\}$$

Copies the element with the least magnitude from real vector `A` to real scalar `C`, and writes its zero-based index to integer scalar `IC`. The index is the actual array index, not the pre-stride index. If vector `A` contains more than one instance of the least magnitude, `IC` contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_minmgviD

Vector minimum magnitude with index; double precision.

```
void vDSP_minmgviD (double * A,
vDSP_Stride I,
double * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A
Double-precision real input vector

I
Stride for `A`

C
Output scalar

IC
Output scalar

N
Count

Discussion
This performs the operation

$$c = |A_0| \quad d = 0 \qquad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \quad d = ni \qquad n = \{1, N-1\}$$

Copies the element with the least magnitude from real vector `A` to real scalar `C`, and writes its zero-based index to integer scalar `IC`. The index is the actual array index, not the pre-stride index. If vector `A` contains more than one instance of the least magnitude, `IC` contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_minv

Vector minimum value.

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

Parameters

A
Single-precision real input vector. If passed an array of length 0, this function returns +INF.

I
Stride for A

C
Output scalar

N
Count

Discussion
This performs the operation

$$c = A_0 \quad \text{If} \quad c > A_{ni} \quad \text{then} \quad c = A_{ni} \qquad \text{n = \{1, N-1\}}$$

Finds the element with the least value in vector A and copies this value to scalar C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_minvD

Vector minimum value; double precision.

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

Parameters

A
Double-precision real input vector

I
: Stride for A

C
: Output scalar

N
: Count

Discussion
This performs the operation

$$c = A_0 \quad \text{If} \quad c > A_{ni} \quad \text{then} \quad c = A_{ni} \qquad n = \{1, N\text{-}1\}$$

Finds the element with the least value in vector A and copies this value to scalar C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_minvi

Vector minimum value with index; single precision.

```
void vDSP_minvi (float * A,
vDSP_Stride I,
float * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A
: Single-precision real input vector. If passed a zero vector, this function returns a value of +INF with an indeterminate index.

I
: Stride for A

C
: Output scalar

IC
: Output scalar index

N
: Count

Discussion
This performs the operation

$$\begin{matrix} c = A_0 \\ d = 0 \end{matrix} \quad \text{If} \quad c > A_{ni} \quad \text{then} \quad \begin{matrix} c = A_{ni} \\ d = ni \end{matrix} \qquad n = \{1, N\text{-}1\}$$

Copies the element with the least value from real vector A to real scalar C, and writes its zero-based index to integer scalar IC. The index is the actual array index, not the pre-stride index. If vector A contains more than one instance of the least value, IC contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_minviD

Vector minimum value with index; double precision.

```
void vDSP_minviD (double * A,
vDSP_Stride I,
double * C,
vDSP_Length * IC,
vDSP_Length N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for A

C
: Output scalar

IC
: Output scalar

N
: Count

Discussion
This performs the operation

$$\begin{matrix} c = A_0 \\ d = 0 \end{matrix} \quad \text{If} \quad c > A_{ni} \quad \text{then} \quad \begin{matrix} c = A_{ni} \\ d = ni \end{matrix} \qquad n = \{1, N-1\}$$

Copies the element with the least value from real vector A to real scalar C, and writes its zero-based index to integer scalar IC. The index is the actual array index, not the pre-stride index. If vector A contains more than one instance of the least value, IC contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_mvessq

Vector mean of signed squares; single precision.

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

Parameters

A
: Single-precision real input vector.

I
: Stride for `A`

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$C = \frac{1}{N}\sum_{n=0}^{N-1} A_{ni} \cdot \left|A_{ni}\right|$$

Finds the mean value of the signed squares of the elements of vector `A` and stores this value in `*C`. If `A` is an array of length 0, this function returns a NaN. .

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_mvessqD

Vector mean of signed squares; double precision.

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

Parameters

A
: Double-precision real input vector

I
: Stride for `A`

C
: Output scalar

N
: Count

Discussion
This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} A_{ni} \cdot |A_{ni}|$$

Finds the mean value of the signed squares of the elements of vector `A` and stores this value in `*C`. If `A` is an array of length 0, this function returns a NaN.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_rmsqv

Vector root-mean-square; single precision.

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

Parameters

A
: Single-precision real input vector. If passed an array of length 0, this function returns a NaN.

I
: Stride for `A`

C
: Single-precision real output scalar

N
: Count

Discussion
This performs the operation

$$C = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} A_{nI}^{2}}$$

Calculates the root mean square of the elements of `A` and stores the result in `*C`

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_rmsqvD

Vector root-mean-square; double precision.

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

Parameters

A
Double-precision real input vector

I
Stride for `A`

C
Double-precision real output scalar

N
Count

Discussion
This performs the operation

$$C = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} A_{nI}^2}$$

Calculates the root mean square of the elements of `A` and stores the result in `*C`

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_sve

Vector sum; single precision.

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

Parameters

A
Single-precision real input vector.

I

Stride for A

C

Single-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}$$

Writes the sum of the elements of A into *C. If A is a vector with zero elements, this function returns 0 in *C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_sveD

Vector sum; double precision.

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

Parameters

A

Double-precision real input vector

I

Stride for A

C

Double-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}$$

Writes the sum of the elements of A into *C. If A is a vector with zero elements, this function returns 0 in *C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_svemg

Vector sum of magnitudes; single precision.

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

Parameters

A
: Single-precision real input scalar. If passed an value of 0, this function returns 0.

I
: Stride for `A`

C
: Single-precision real output scalar

N
: Count

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} |A_{nI}|$$

Writes the sum of the magnitudes of the elements of `A` into `C`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_svemgD

Vector sum of magnitudes; double precision.

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

Parameters

A

Double-precision real input scalar

I

Stride for A

C

Double-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} |A_{nI}|$$

Writes the sum of the magnitudes of the elements of A into C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_svesq

Vector sum of squares; single precision.

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

Parameters

A

Single-precision real input scalar. If passed an value of 0, this function returns 0.

I

Stride for A

C

Single-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}^{2}$$

Writes the sum of the squares of the elements of A into C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_svesqD

Vector sum of squares; double precision.

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

Parameters

A
Double-precision real input scalar

I
Stride for A

C
Double-precision real output scalar

N
Count

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}^{2}$$

Writes the sum of the squares of the elements of A into C.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_svs

Vector sum of signed squares; single precision.

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

Parameters

A

Single-precision real input scalar. If passed an value of 0, this function returns 0.

I

Stride for A

C

Single-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \cdot |A_{nI}|$$

Writes the sum of the signed squares of the elements of A into C.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_svsD

Vector sum of signed squares; double precision.

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

Parameters

A

Double-precision real input scalar

I

Stride for A

C

Double-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \bullet |A_{nI}|$$

Writes the sum of the signed squares of the elements of `A` into `C`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zdotpr

Calculates the complex dot product of complex vectors signal1 and signal2 and leaves the result in complex vector `result`; single precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zdotprD

Calculates the complex dot product of complex vectors signal1 and signal2 and leaves the result in complex vector `result`; double precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zidotpr

Calculates the conjugate dot product (or inner dot product) of complex vectors `signal1` and `signal2` and leave the result in complex vector `result`; single precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}^{*} B_{nJ}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zidotprD

Calculates the conjugate dot product (or inner dot product) of complex vectors signal1 and signal2 and leave the result in complex vector `result`; double precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A^*_{nI} B_{nJ}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zrdotpr

Calculates the complex dot product of complex vector `A` and real vector `B` and leaves the result in complex vector `C`; single precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zrdotprD

Calculates the complex dot product of complex vector `A` and real vector `B` and leaves the result in complex vector `C`; double precision.

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

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

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 166)
Vector equivalence, 32-bit logical.

Doing Basic Arithmetic on Real Vectors

`vDSP_vadd` (page 152)
Adds vector `A` to vector `B` and leaves the result in vector `C`; single precision.

`vDSP_vaddD` (page 153)
Adds vector `A` to vector `B` and leaves the result in vector `C`; double precision.

`vDSP_vsub` (page 201)
Subtracts vector `signal1` from vector `signal2` and leaves the result in vector `result`; single precision.

`vDSP_vsubD` (page 201)
Subtracts vector `signal1` from vector `signal2` and leaves the result in vector `result`; double precision.

`vDSP_vam` (page 153)
Adds vectors `A` and `B`, multiplies the sum by vector `C`, and leaves the result in vector `D`; single precision.

`vDSP_vamD` (page 154)
Adds vectors `A` and `B`, multiplies the sum by vector `C`, and leaves the result in vector `D`; double precision.

`vDSP_vsbm` (page 195)
Vector subtract and multiply; single precision.

Doing Basic Arithmetic on Complex Vectors

Finding Maximum and Minimum Elements

Computing Vector Distance

Interpolating Between Two Vectors

Evaluating Vectors as Polynomials

Applying Pythagoras's Theorem to Vector Elements

Finding a Vector's Extrema

Swapping Elements Between Vectors

Merging Two Vectors

Computing Vector Spectra

Computing the Coherence Function of Two Vectors

Computing the Transfer Function

Doing Recursive Filtering on a Real Vector

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

A

Single-precision real input vector; must have at least N+2 elements

I

Stride for A

B

5 single-precision inputs, filter coefficients

C

Single-precision real output vector; must have at least N+2 elements

K

Stride for C

N

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.

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

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

A

Double-precision real input vector; must have at least N+2 elements

I

Stride for A

B

5 double-precision inputs, filter coefficients

C

Double-precision real output vector; must have at least N+2 elements

K

Stride for C

N

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.

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

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

A

Single-precision real input vector

I

Stride for A

B

Single-precision real input vector

J
  Stride for B

C
  Single-precision real input vector

K
  Stride for C

D
  Single-precision real input vector

L
  Stride for D

E
  Single-precision real output vector

M
  Stride for E

N
  Count ; each vector must have at least N elements

Discussion

This performs the operation

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

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
: Stride for A

B
: Double-precision real input vector

J
: Stride for B

C
: Double-precision real input vector

K
: Stride for C

D
: Double-precision real input vector

L
: Stride for D

E
: Double-precision real output vector

M
: Stride for E

N
: Count ; each vector must have at least N elements

Discussion

This performs the operation

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

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

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

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

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

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

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

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

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

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`

B
: Single-precision real input vector

J
: Stride for `B`

C
: Single-precision real input vector

K
: Stride for `C`

D
: Single-precision real input vector

L
: Stride for `D`

E
: Single-precision real output vector

M
: Stride for `E`

N
: Count ; each vector must have at least `N` elements

Discussion

This performs the operation

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

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

A
: Double-precision real input vector

I
: Stride for A

B
: Double-precision real input vector

J
: Stride for B

C
: Double-precision real input vector

K
: Stride for C

D
: Double-precision real input vector

L
: Stride for D

E
: Double-precision real output vector

M
: Stride for E

N
: Count ; each vector must have at least N elements

Discussion

This performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for `A`

B
: Single-precision real input vector

J
: Stride for `B`

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for `D`

N
: Count ; each vector must have at least N elements

Discussion

This performs the operation

$$D_{nM} = (A_{nI} + B_{nJ})\,c \;, \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

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

A
: Double-precision real input vector

I
: Stride for A

B
: Double-precision real input vector

J
: Stride for B

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for D

N
: Count ; each vector must have at least N elements

Discussion

This performs the operation

$$D_{nM} = (A_{nI} + B_{nJ})\,c\ , \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

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

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

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

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

A
: Double-precision real 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

Performs the operation

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

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

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

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

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

A
: Double-precision real 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

Performs the operation

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

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

A
Integer input vector

I
Stride for A

B
Integer input vector

J
Stride for B

C
Integer output vector

K
Stride for C

N
Count

Discussion

Performs the operation

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

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

A
: Single-precision real input vector: high envelope

I
: Stride for `A`

B
: Single-precision real input vector: low envelope

J
: Stride for `B`

C
: Single-precision real input vector

K
: Stride for `C`

D
: Single-precision real output vector

L
: Stride for `D`

N
: Count

Discussion

Performs the operation

$$\text{If } C_{nK} > A_{nI} \text{ or } C_{nK} < B_{nJ} \text{ then } D_{nM} = C_{nK}$$
$$\text{else } D_{nM} = 0.0 \qquad n = \{0, N\text{-}1\}$$

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

A
: Double-precision real input vector: high envelope

I
: Stride for `A`

B
: Double-precision real input vector: low envelope

J
: Stride for `B`

C
: Double-precision real input vector

K
: Stride for `C`

D
: Double-precision real output vector

L
: Stride for `D`

N
: Count

Discussion

Performs the operation

$$\text{If } C_{nK} > A_{nI} \text{ or } C_{nK} < B_{nJ} \text{ then } D_{nM} = C_{nK}$$
$$\text{else } D_{nM} = 0.0 \qquad n = \{0, N\text{-}1\}$$

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

A
: Integer input vector

I
: Stride for `A`

B
: Integer input vector

J
: Stride for `B`

C
: Integer output vector

K
: Stride for `C`

N
: Count

Discussion

Performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for `A`

B
: Single-precision real input vector

J
: Stride for `B`

C
: Single-precision real input scalar: interpolation constant

D
: Single-precision real output vector

L
: Stride for `D`

N
: Count

Discussion
Performs the operation

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

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

A
: Double-precision real input vector

I
: Stride for `A`

B
: Double-precision real input vector

J
: Stride for `B`

C
: Double-precision real input scalar: interpolation constant

D
: Double-precision real output vector

L
: Stride for `D`

N
: Count

Discussion

Performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for A

B
: Single-precision real input vector

J
: Stride for B

C
: Single-precision real input vector

K
: Stride for C

D
: Single-precision real output vector

L
: Stride for D

N
: Count

Discussion

This performs the operation

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

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

A
Double-precision real input vector

I
Stride for A

B
Double-precision real input vector

J
Stride for B

C
Double-precision real input vector

K
Stride for C

D
Double-precision real output vector

L
Stride for D

N
Count

Discussion
This performs the operation

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

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

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

This performs the operation

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

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

A
: Double-precision real 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

This performs the operation

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

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

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

This performs the operation

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

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

A
: Double-precision real 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

This performs the operation

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

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

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

This performs the operation

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

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

A
Double-precision real 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

This performs the operation

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

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

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

This performs the operation

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

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

A
: Double-precision real 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

This performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for A

B
: Single-precision real input vector

J
: Stride for B

C
: Single-precision real input vector

K
: Stride for C

D
: Single-precision real input vector

L
: Stride for D

E
: Single-precision real output vector

M
: Stride for E

N
: Count

Discussion

This performs the operation

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

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

A
Double-precision real input vector

I
Stride for A

B
Double-precision real input vector

J
Stride for B

C
Double-precision real input vector

K
Stride for C

D
Double-precision real input vector

L
Stride for D

E
Double-precision real output vector

M
Stride for E

N
Count

Discussion

This performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for `A`

B
: Single-precision real input vector

J
: Stride for `B`

C
: Single-precision real input vector

K
: Stride for `C`

D
: Single-precision real input vector

L
: Stride for `D`

E
: Single-precision real output vector

M
: Stride for `E`

N
: Count

Discussion
This performs the operation

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

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

A
: Double-precision real input vector

I
: Stride for `A`

B
: Double-precision real input vector

J
: Stride for `B`

C
: Double-precision real input vector

K
: Stride for `C`

D
: Double-precision real input vector

L
: Stride for `D`

E
: Double-precision real output vector

M
: Stride for `E`

N
: Count

Discussion
This performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for `A`

B
: Single-precision real input vector

J
: Stride for `B`

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for `D`

N
: Count

Discussion
This performs the operation

$$D_{nK} = A_{nI} \bullet B_{nJ} + C \qquad n = \{0, N\text{-}1\}$$

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

A
: Double-precision real input vector

I
: Stride for A

B
: Double-precision real input vector

J
: Stride for B

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for D

N
: Count

Discussion
This performs the operation

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

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

A
    Single-precision real input vector

I
    Stride for A

B
    Single-precision real input vector

J
    Stride for B

C
    Single-precision real input vector

K
    Stride for C

D
    Single-precision real output vector

L
    Stride for D

N
    Count

Discussion

This performs the operation

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

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

A
: Double-precision real input vector

I
: Stride for A

B
: Double-precision real input vector

J
: Stride for B

C
: Double-precision real input vector

K
: Stride for C

D
: Double-precision real output vector

L
: Stride for D

N
: Count

Discussion

This performs the operation

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

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

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

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

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

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

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

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

A

Single-precision real input vector: coefficients

I
    Stride for A

B
    Single-precision real input vector: variable values

J
    Stride for B

C
    Single-precision real output vector

K
    Stride for C

N
    Count

P
    Degree of polynomial

Discussion

Performs the operation

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

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

A
    Double-precision real input vector: coefficients

I
    Stride for A

B
: Double-precision real input vector: variable values

J
: Stride for `B`

C
: Double-precision real output vector

K
: Stride for `C`

N
: Count

P
: Degree of polynomial

Discussion

Performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for `A`

B
Single-precision real input vector

J
Stride for B

C
Single-precision real input vector

K
Stride for C

D
Single-precision real input vector

L
Stride for D

E
Single-precision real output vector

M
Stride for E

N
Count

Discussion
Performs the operation

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

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

A
Double-precision real input vector

I
Stride for A

B
Double-precision real input vector

J
Stride for B

C
Double-precision real input vector

K
Stride for C

D
Double-precision real input vector

L
Stride for D

E
Double-precision real output vector

M
Stride for E

N
Count

Discussion

Performs the operation

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

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

A
: Single-precision real input vector

B
: Single-precision real input vector: integer parts are indices into A and fractional parts are interpolation constants

J
: Stride for B

C
: Single-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

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

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

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

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

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

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

A
Single-precision real input vector

I
Stride for `A`

B
Single-precision real input vector

J
Stride for `B`

C
Single-precision real input vector

K
Stride for `C`

D
Single-precision real output vector

L
Stride for `D`

N
Count

Discussion
Performs the operation

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

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_vsbmD

Vector subtract and multiply; double precision.

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

Parameters

A
: Double-precision real input vector

I
: Stride for `A`

B
: Double-precision real input vector

J
: Double for `B`

C
: Double-precision real input vector

K
: Stride for `C`

D
: Double-precision real output vector

L
: Stride for `D`

N
: Count

Discussion
Performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for A

B
: Single-precision real input vector

J
: Stride for B

C
: Single-precision real input vector

K
: Stride for C

D
: Single-precision real input vector

L
: Stride for D

E
: Single-precision real output vector

M
: Stride for E

N
: Count

Discussion

Performs the operation

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

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

A
: Double-precision real input vector

I
: Stride for `A`

B
: Double-precision real input vector

J
: Stride for `B`

C
: Double-precision real input vector

K
: Stride for `C`

D
: Double-precision real input vector

L
: Stride for `D`

E
: Double-precision real output vector

M
: Stride for `E`

N
: Count

Discussion

Performs the operation

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

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

A
: Single-precision real input vector

I
: Stride for A

B
: Single-precision real input vector

J
: Stride for B

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for D

N
: Count

Discussion
Performs the operation

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

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

A
Double-precision real input vector

I
Stride for `A`

B
Double-precision real input vector

J
Stride for `B`

C
Double-precision real input scalar

D
Double-precision real output vector

L
Stride for `D`

N
Count

Discussion
Performs the operation

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

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

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

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

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

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

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

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

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

Exchanges the elements of vectors A and B.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_vtmerg

Tapered merge of two vectors; single precision.

```
void vDSP_vtmerg (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

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

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

A
Double-precision real 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
Performs the operation

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

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*x + y*y). 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`.

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

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*x + y*y). 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`.

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

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`.

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

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`.

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

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`.

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

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

A
: Double-precision complex input vector

B
: Double-precision complex input vector

C
: Double-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_zcspecD.

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

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

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

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

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

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

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zrvdiv

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

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

Discussion
This performs the operation

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

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zrvdivD

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

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

Discussion
This peforms the operation

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

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

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 peforms the operation

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

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

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

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

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

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

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

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zrvsubD

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

```
void vDSP_zrvsubD (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 peforms the operation

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

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 peforms the operation

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

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

A
: Double-precision real input vector

B
: Double-precision complex input vector

C
: Double-precision complex output vector

N
: Count

Discussion

This peforms the operation

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

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

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 peforms the operation

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

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 peforms the operation

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

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 peforms the operation

$$D_{nL} = A^{*}_{nI} B_{nJ} + C_{nK} \qquad n = \{0, N\text{-}1\}$$

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 peforms the operation

$$D_{nL} = A^{*}_{nI} B_{nJ} + C_{nK} \qquad n = \{0, N\text{-}1\}$$

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

A
: Single-precision complex input vector

I
: Stride for A

B
: Single-precision complex input vector

J
: Stride for B

B
: Single-precision complex output vector

K
: Stride for B

N
: 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

A
: Double-precision complex input vector

I
: Stride for A

B
: Double-precision complex input vector

J
: Stride for B

B
: Double-precision complex output vector

K
: Stride for B

N
: 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.

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

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

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

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 peforms the operation

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

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 peforms the operation

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

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP Matrix Operations Reference

Framework: Accelerate/vecLib

Declared in vDSP.h

Overview

This document describes the C API for the matrix arithmetic operations available in vDSP. It provides functionality for multiplying and transposing real or complex matrices.

Functions by Task

Multiplying Real Matrices

Transposing a Matrix

Copying a Submatrix

Multiplying Complex Matrices

Functions

vDSP_mmov

The contents of a submatrix are copied to another submatrix.

```
void
vDSP_mmov (float * A,
float * C,
vDSP_Length NC,
vDSP_Length NR,
vDSP_Length TCA,
vDSP_Length TCC);
```

Parameters

A

Single-precision real input submatrix

C

Single-precision real output submatrix

NC
Number of columns in A and C

NR
Number of rows in A and C

TCA
Number of columns in the matrix of which A is a submatrix

TCC
Number of columns in the matrix of which C is a submatrix

Discussion

The matrices are assumed to be stored in row-major order. Thus elements A[i][j] and A[i][j+1] are adjacent. Elements A[i][j] and A[i+1][j] are TCA elements apart.

This function may be used to move a subarray beginning at any point in a larger embedding array by passing for A the address of the first element of the subarray. For example, to move a subarray starting at A[3][4], pass &A[3][4]. Similarly, the address of the first destination element is passed for C

NC may equal TCA, and it may equal TCC. To copy all of an array to all of another array, pass the number of rows in NR and the number of columns in NC, TCA, and TCC.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_mmovD

The contents of a submatrix are copied to another submatrix.

```
void vDSP_mmovD (double * A,
double * C,
vDSP_Length NC,
vDSP_Length NR,
vDSP_Length TCA,
vDSP_Length TCC);
```

Parameters

A
Double-precision real input submatrix

C
Double-precision real output submatrix

NC
Number of columns in A and C

NR
Number of rows in A and C

TCA
Number of columns in the matrix of which A is a submatrix

TCC
Number of columns in the matrix of which C is a submatrix

Discussion

The matrices are assumed to be stored in row-major order. Thus elements `A[i][j]` and `A[i][j+1]` are adjacent. Elements `A[i][j]` and `A[i+1][j]` are `TCA` elements apart.

This function may be used to move a subarray beginning at any point in a larger embedding array by passing for `A` the address of the first element of the subarray. For example, to move a subarray starting at `A[3][4]`, pass `&A[3][4]`. Similarly, the address of the first destination element is passed for `C`

`NC` may equal `TCA`, and it may equal `TCC`. To copy all of an array to all of another array, pass the number of rows in `NR` and the number of columns in `NC`, `TCA`, and `TCC`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_mmul

Performs an out-of-place multiplication of `M`-by-`P` matrix `A` by a `P`-by-`N` matrix `B` and stores the results in an `M`-by-`N` matrix `C`; single precision.

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

Discussion
This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \cdot B_{(pN+n)J} \qquad n = \{0, N-1\} \text{ and } m = \{0, M-1\}$$

Parameters `A` and `B` are the matrixes to be multiplied. `I` is an address stride through `A`. `J` is an address stride through `B`.

Parameter `C` is the result matrix. `K` is an address stride through `C`.

Parameter `M` is the row count for both `A` and `C`. Parameter `N` is the column count for both `B` and `C`. Parameter `P` is the column count for `A` and the row count for `B`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_mmulD

Performs an out-of-place multiplication of M-by-P matrix A by a P-by-N matrix B and stores the results in an M-by-N matrix C; double precision.

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

Discussion
This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \cdot B_{(pN+n)J} \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

Parameters A and B are the matrixes to be multiplied. I is an address stride through A. J is an address stride through B.

Parameter C is the result matrix. K is an address stride through C.

Parameter M is the row count for both A and C. Parameter N is the column count for both B and C. Parameter P is the column count for A and the row count for B.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_mtrans

Creates a transposed matrix C from a source matrix A; single precision.

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

Discussion
This performs the operation

$$C_{(mN+n)K} = A_{(nM+m)I} \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

Parameter A is the source matrix. I is an address stride through the source matrix.

Parameter C is the resulting transposed matrix. K is an address stride through the result matrix.

Parameter M is the number of rows in C (and the number of columns in A).

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_mtransD

Creates a transposed matrix C from a source matrix A; double precision.

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

Discussion
This performs the operation

$$C_{(mN+n)K} = A_{(nM+m)I} \qquad \text{n = \{0, N-1\} and m = \{0, M-1\}}$$

Parameter A is the source matrix. I is an address stride through the source matrix.

Parameter C is the resulting transposed matrix. K is an address stride through the result matrix.

Parameter M is the number of rows in C (and the number of columns in A).

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_zmma

Performs an out-of-place complex multiplication of an M-by-P matrix A by a P-by-N matrix B, adds the product to M-by-N matrix C, and stores the result in M-by-N matrix D; single precision.

```
void vDSP_zmma (DSPSplitComplex * A,
vDSP_Stride I,
DSPSplitComplex * B,
vDSP_Stride J,
DSPSplitComplex * C,
vDSP_Stride K,
DSPSplitComplex * D,
vDSP_Stride L,
vDSP_Length M,
vDSP_Length N,
vDSP_Length P);
```

Discussion

This performs the operation

$$D_{(rN+q)L} = C_{(rN+q)K} + \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J}$$

$$0 \le r < M , \quad 0 \le q < N$$

Parameters A and C are the matrixes to be multiplied, and C the matrix to be added. I is an address stride through A. J is an address stride through B. K is an address stride through C. L is an address stride through D.

Parameter D is the result matrix.

Parameter M is the row count for A, C and D. Parameter N is the column count of B, C, and D. Parameter P is the column count of A and the row count of B.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zmmaD

Performs an out-of-place complex multiplication of an M-by-P matrix A by a P-by-N matrix B, adds the product to M-by-N matrix C, and stores the result in M-by-N matrix D; double precision.

```
void vDSP_zmmaD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
DSPDoubleSplitComplex * B,
vDSP_Stride J,
DSPDoubleSplitComplex * C,
vDSP_Stride K,
DSPDoubleSplitComplex * D,
vDSP_Stride L,
vDSP_Length M,
vDSP_Length N,
vDSP_Length P);
```

Discussion

This performs the operation

$$D_{(rN+q)L} = C_{(rN+q)K} + \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J}$$

$$0 \le r < M, \quad 0 \le q < N$$

Parameters A and C are the matrixes to be multiplied, and C the matrix to be added. I is an address stride through A. J is an address stride through B. K is an address stride through C. L is an address stride through D.

Parameter D is the result matrix.

Parameter M is the row count for A, C and D. Parameter N is the column count of B, C, and D. Parameter P is the column count of A and the row count of B.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zmms

Performs an out-of-place complex multiplication of an M-by-P matrix A by a P-by-N matrix B , subtracts M-by-N matrix C from the product, and stores the result in M-by-N matrix D; single precision.

```
void vDSP_zmms (DSPSplitComplex * A,
vDSP_Stride I,
DSPSplitComplex * B,
vDSP_Stride J,
DSPSplitComplex * C,
vDSP_Stride K,
DSPSplitComplex * D,
vDSP_Stride L,
vDSP_Length M,
vDSP_Length N,
vDSP_Length P);
```

Discussion

This performs the operation

$$D_{(rN+q)L} = \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J} - C_{(rN+q)K}$$

$$0 \le r < M, \quad 0 \le q < N$$

Parameters A and B are the matrixes to be multiplied, and C the matrix to be subtracted. I is an address stride through A. J is an address stride through B. K is an address stride through C. L is an address stride through D.

Parameter D is the result matrix.

Parameter M is the row count for A, C and D. Parameter N is the column count of B, C, and D. Parameter P is the column count of A and the row count of B.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_zmmsD

Performs an out-of-place complex multiplication of an M-by-P matrix A by a P-by-N matrix B , subtracts M-by-N matrix C from the product, and stores the result in M-by-N matrix D; double precision.

```
void vDSP_zmmsD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
DSPDoubleSplitComplex * B,
vDSP_Stride J,
DSPDoubleSplitComplex * C,
vDSP_Stride K,
DSPDoubleSplitComplex * D,
vDSP_Stride L,
vDSP_Length M,
vDSP_Length N,
vDSP_Length P);
```

Discussion
This performs the operation

$$D_{(rN+q)L} = \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J} - C_{(rN+q)K}$$

$$0 \le r < M, \quad 0 \le q < N$$

Parameters A and B are the matrixes to be multiplied, and C the matrix to be subtracted. I is an address stride through A. J is an address stride through B. K is an address stride through C. L is an address stride through D.

Parameter D is the result matrix.

Parameter M is the row count for A, C and D. Parameter N is the column count of B, C, and D. Parameter P is the column count of A and the row count of B.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_zmmul

Performs an out-of-place complex multiplication of an M-by-P matrix A by a P-by-N matrix B and stores the results in an M-by-N matrix C; single precision.

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

Discussion
This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \cdot B_{(pN+n)J} \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

Parameters A and B are the matrixes to be multiplied. I is an address stride through A. J is an address stride through B.

Parameter C is the result matrix. K is an address stride through C.

Parameter M is the row count for both A and C. Parameter N is the column count for both B and C. Parameter P is the column count for A and the row count for B.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_zmmulD

Performs an out-of-place complex multiplication of an M-by-P matrix A by a P-by-N matrix B and stores the results in an M-by-N matrix C; double precision.

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

Discussion

This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \cdot B_{(pN+n)J} \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

Parameters `A` and `B` are the matrixes to be multiplied. `I` is an address stride through `A`. `J` is an address stride through `B`.

Parameter `C` is the result matrix. `K` is an address stride through `C`.

Parameter `M` is the row count for both `A` and `C`. Parameter `N` is the column count for both `B` and `C`. Parameter `P` is the column count for `A` and the row count for `B`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_zmsm

Performs an out-of-place complex multiplication of an `M`-by-`P` matrix `A` by a `P`-by-`N` matrix `B`, subtracts the product from `M`-by-`P` matrix `C`, and stores the result in `M`-by-`P` matrix *D*; single precision.

```
void vDSP_zmsm (DSPSplitComplex * A,
vDSP_Stride I,
DSPSplitComplex * B,
vDSP_Stride J,
DSPSplitComplex * C,
vDSP_Stride K,
DSPSplitComplex * D,
vDSP_Stride L,
vDSP_Length M,
vDSP_Length N,
vDSP_Length P);
```

Discussion

This performs the operation

$$D_{(rN+q)L} = C_{(rN+q)K} - \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J}$$

Parameters `A` and `B` are the matrixes to be multiplied, and `C` is the matrix from which the product is to be subtracted. `aStride` is an address stride through `A`. `bStride` is an address stride through `B`. `cStride` is an address stride through `C`. `dStride` is an address stride through `D`.

Parameter `D` is the result matrix.

Parameter `M` is the row count for `A`, `C` and `D`. Parameter `N` is the column count of `B`, `C`, and `D`. Parameter `P` is the column count of `A` and the row count of `B`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zmsmD

Performs an out-of-place complex multiplication of an `M`-by-`P` matrix `A` by a `P`-by-`N` matrix `B`, subtracts the product from `M`-by-`P` matrix `C`, and stores the result in `M`-by-`P` matrix `D`; double precision.

```
void vDSP_zmsmD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
DSPDoubleSplitComplex * B,
vDSP_Stride J,
DSPDoubleSplitComplex * C,
vDSP_Stride K,
DSPDoubleSplitComplex * D,
vDSP_Stride L,
vDSP_Length M,
vDSP_Length N,
vDSP_Length P);
```

Discussion
This performs the operation

$$D_{(rN+q)L} = C_{(rN+q)K} - \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J}$$

Parameters `A` and `B` are the matrixes to be multiplied, and parameter `C` is the matrix from which the product is to be subtracted. `aStride` is an address stride through `A`. `bStride` is an address stride through `B`. `cStride` is an address stride through `C`. `dStride` is an address stride through `D`.

Parameter `D` is the result matrix.

Parameter `M` is the row count for `A`, `C` and `D`. Parameter `N` is the column count of `B`, `C`, and `D`. Parameter `P` is the column count of `A` and the row count of `B`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP Correlation, Convolution, and Filtering Reference

Framework: Accelerate/vecLib

Declared in vDSP.h

Overview

This document describes the C API for performing correlation, convolution, and filtering operations on real or complex signals in vDSP. It also describes the built-in support for windowing functions such as Blackman, Hamming, and Hann windows.

Functions by Task

Correlation and Convolution

Windowing and Filtering

Functions

vDSP_blkman_window

Creates a single-precision Blackman window.

```
void vDSP_blkman_window(
float * C,
vDSP_Length N,
int FLAG);
```

Discussion
Represented in pseudo-code, this function does the following:

```
for (n=0; n < N; ++n)
```

```
{
    C[n] = 0.42 - (0.5 * cos(  2 * pi * n / N ) ) + (0.08 * cos( 4 * pi * n /
N) );
}
```

vDSP_blkman_window creates a single-precision Blackman window function C, which can be multiplied by a vector using vDSP_vmul. Specify the vDSP_HALF_WINDOW flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also vDSP_vmul.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_blkman_windowD

Creates a double-precision Blackman window.

```
void vDSP_blkman_windowD (double * C,
vDSP_Length N,
int FLAG);
```

Discussion
Represented in pseudo-code, this function does the following:

```
for (n=0; n < N; ++n)
{
    C[n] = 0.42 - (0.5 * cos(  2 * pi * n / N ) ) + (0.08 * cos( 4 * pi * n /
N) );
}
```

vDSP_blkman_windowD (page 239) creates a double-precision Blackman window function C, which can be multiplied by a vector using vDSP_vmulD . Specify the vDSP_HALF_WINDOW flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also vDSP_vmulD.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_conv

Performs either correlation or convolution on two vectors; single precision.

```
vDSP_conv (const float signal[],
vDSP_Stride signalStride,
const float filter[],
vDSP_Stride strideFilter,
float result[],
vDSP_Stride strideResult,
vDSP_Length lenResult,
vDSP_Length lenFilter);
```

Discussion

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad \text{n = \{0, N-1\}}$$

If `filterStride` is positive, `vDSP_conv` performs correlation. If `filterStride` is negative,it performs convolution and *filtermust point to the last vector element. The function can run in place, but result cannot be in place with filter.

The value of `lenFilter` must be less than or equal to 2044. The length of vector `signal` must satisfy two criteria: it must be

- equal to or greater than 12
- equal to or greater than the sum of `N-1` plus the nearest multiple of 4 that is equal to or greater than the value of `lenFilter`.

Criteria to invoke vectorized code:

- The vectors `signal` and `result` must be relatively aligned.
- The value of `lenFilter` must be between 4 and 256, inclusive.
- The value of `lenResult` must be greater than 36.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `filterStride` must be either 1 or -1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_convD

Performs either correlation or convolution on two vectors; double precision.

```
void vDSP_convD (const double signal[],
vDSP_Stride signalStride,
const double filter[],
vDSP_Stride strideFilter,
double result[],
vDSP_Stride strideResult,
vDSP_Length lenResult,
vDSP_Length lenFilter);
```

Discussion

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad \mathrm{n} = \{0, \mathrm{N\text{-}1}\}$$

If `filterStride` is positive, `vDSP_convD` performs correlation. If `filterStride` is negative,it performs convolution and `*filter`must point to the last vector element. The function can run in place, but result cannot be in place with filter.

The value of `lenFilter` must be less than or equal to 2044. The length of vector `signal` must satisfy two criteria: it must be

- equal to or greater than 12
- equal to or greater than the sum of `N-1` plus the nearest multiple of 4 that is equal to or greater than the value of `lenFilter`.

Criteria to invoke vectorized code:

No Altivec support for double precision. On a PowerPC processor, this function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_desamp

Convolution with decimation; single precision.

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

Parameters

A

Single-precision real input vector, 8-byte aligned; length of `A` >= 12

I

Desampling factor

B

Single-precision input filter coefficients

C

Single-precision real output vector

N

Output count

M

Filter coefficient count

Discussion

Performs finite impulse response (FIR) filtering at selected positions of vector `A`. `desampx` can run in place, but `C` cannot be in place with `B`. Length of `A` must be >=(`N`-1)*`I`+(nearest multiple of 4 >=`M`).

$$C_n = \sum_{p=0}^{P-1} A_{nI+p} B_p \qquad \text{n = \{0, N-1\}}$$

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_desampD

Convolution with decimation; double precision.

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

Parameters

A

Double-precision real input vector, 8-byte aligned; length of `A` >= 12

I

Desampling factor

B

Double-precision input filter coefficients

C

Double-precision real output vector

N

Output count

M

Filter coefficient count

Discussion
Performs finite impulse response (FIR) filtering at selected positions of vector A. desampx can run in place, but C cannot be in place with B. Length of A must be >=(N-1)*I+(nearest multiple of 4 >=M).

$$C_n = \sum_{p=0}^{P-1} A_{nI+p} B_p \qquad \text{n = \{0, N-1\}}$$

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_f3x3

Filters an image by performing a two-dimensional convolution with a 3x3 kernel on the input matrix A. The resulting image is placed in the output matrix C; single precision.

```
void vDSP_f3x3 (float * signal,
vDSP_Length rows,
vDSP_Length cols,
float * filter,
float * result);
```

Discussion
This performs the operation

$$C_{(m+1,n+1)} = \sum_{p=0}^{2} \sum_{q=0}^{2} A_{(m+p,n+q)} \cdot B_{(p,q)} \qquad \text{m = \{0, M-1\} and n = \{0, N-3\}}$$

The function pads the perimeter of the output image with a border of zeros of width 1.

B is the 3x3 kernel. M and N are the number of rows and columns, respectively, of the two-dimensional input matrix A. M must be greater than or equal to 3. N must be even and greater than or equal to 4.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_f3x3D

Filters an image by performing a two-dimensional convolution with a 3x3 kernel on the input matrix A. The resulting image is placed in the output matrix C; double precision.

```
void vDSP_f3x3D (double * signal,
vDSP_Length rows,
vDSP_Length cols,
double * filter,
double * result);
```

Discussion

This performs the operation

$$C_{(m+1,n+1)} = \sum_{p=0}^{2} \sum_{q=0}^{2} A_{(m+p,n+q)} \bullet B_{(p,q)} \qquad m = \{0, M-1\} \text{ and } n = \{0, N-3\}$$

The function pads the perimeter of the output image with a border of zeros of width 1.

`B` is the 3x3 kernel. `M` and `N` are the number of rows and columns, respectively, of the two-dimensional input matrix `A`. `M` must be greater than or equal to 3. `N` must be even and greater than or equal to 4.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.
- `N` must be greater than or equal to 18.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_f5x5

Filters an image by performing a two-dimensional convolution with a 5x5 kernel on the input matrix `signal`. The resulting image is placed in the output matrix `result`; single precision.

```
void vDSP_f5x5 (float * A,
vDSP_Length M,
vDSP_Length N,
float * B,
float * C);
```

Discussion

This performs the operation

$$C_{(m+2,n+2)} = \sum_{p=0}^{4} \sum_{q=0}^{4} A_{(m+p,n+q)} \bullet B_{(p,q)} \qquad m = \{0, M-5\} \text{ and } n = \{0, N-5\}$$

The function pads the perimeter of the output image with a border of zeros of width 2.

B is the 3x3 kernel. M and N are the number of rows and columns, respectively, of the two-dimensional input matrix A. M must be greater than or equal to 5. N must be even and greater than or equal to 6.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_f5x5D

Filters an image by performing a two-dimensional convolution with a 5x5 kernel on the input matrix signal. The resulting image is placed in the output matrix result; double precision.

```
void vDSP_f5x5D (double * A,
vDSP_Length M,
vDSP_Length N,
double * B,
double * C);
```

Discussion
This performs the operation

$$C_{(m+2,n+2)} = \sum_{p=0}^{4} \sum_{q=0}^{4} A_{(m+p,n+q)} \cdot B_{(p,q)} \qquad \text{m = \{0, M-5\} and n = \{0, N-5\}}$$

The function pads the perimeter of the output image with a border of zeros of width 2.

B is the 3x3 kernel. M and N are the number of rows and columns, respectively, of the two-dimensional input matrix A. M must be greater than or equal to 5. N must be even and greater than or equal to 6.

Criteria to invoke vectorized code:

- A, B, and C must be 16-byte aligned.
- N must be greater than or equal to 20.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_hamm_window

Creates a single-precision Hamming window.

```
void
vDSP_hamm_window (float * C,
vDSP_Length N,
int FLAG);
```

Discussion

$$C_n = 0.54 - 0.46 \cos \frac{2\pi n}{N} \qquad n = \{0, N\text{-}1\}$$

`vDSP_hamm_window` creates a single-precision Hamming window function `C`, which can be multiplied by a vector using `vDSP_vmul` . Specify the `vDSP_HALF_WINDOW` flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also `vDSP_vmul`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_hamm_windowD

Creates a double-precision Hamming window.

```
void
vDSP_hamm_windowD (double * C,
vDSP_Length N,
int FLAG);
```

Discussion

$$C_n = 0.54 - 0.46 \cos \frac{2\pi n}{N} \qquad n = \{0, N\text{-}1\}$$

`vDSP_hamm_windowD` creates a double-precision Hamming window function `C`, which can be multiplied by a vector using `vDSP_vmulD` . Specify the `vDSP_HALF_WINDOW` flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also `vDSP_vmulD`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_hann_window

Creates a single-precision Hanning window.

```
void
vDSP_hann_window (float * C,
vDSP_Length N,
int FLAG);
```

Discussion

$$C_n = W\left(1.0 - \cos\frac{2\pi n}{N}\right) \qquad n = \{0, N\text{-}1\}$$

`vDSP_hann_window` creates a single-precision Hanning window function `C`, which can be multiplied by a vector using `vDSP_vmul` .

The FLAG parameter can have the following values:

- `vDSP_HANN_DENORM` creates a denormalized window.
- `vDSP_HANN_NORM` creates a normalized window.
- `vDSP_HALF_WINDOW` creates only the first (N+1)/2 points.

`vDSP_HALF_WINDOW` can be ORed with any of the other values (i.e., using the C operator |).

See also `vDSP_vmul`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_hann_windowD

Creates a double-precision Hanning window.

```
void
vDSP_hann_windowD (double * C,
vDSP_Length N,
int FLAG);
```

Discussion

$$C_n = W\left(1.0 - \cos\frac{2\pi n}{N}\right) \qquad n = \{0, N\text{-}1\}$$

`vDSP_hann_window` creates a double-precision Hanning window function `C`, which can be multiplied by a vector using `vDSP_vmul` .

The FLAG parameter can have the following values:

- `vDSP_HANN_DENORM` creates a denormalized window.
- `vDSP_HANN_NORM` creates a normalized window.
- `vDSP_HALF_WINDOW` creates only the first (N+1)/2 points.

vDSP_HALF_WINDOW can ORed with any of the other values (i.e., using the C operator |).

See also vDSP_vmul.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_imgfir

Filters an image by performing a two-dimensional convolution with a kernel; single precision.

```
void vDSP_imgfir (float * A,
vDSP_Length M,
vDSP_Length N,
float * B,
float * C,
vDSP_Length P,
vDSP_Length Q);
```

Parameters

A
: A real matrix signal input.

M
: Number of rows in A.

N
: Number of columns in A.

B
: A two-dimensional real matrix containing the filter.

C
: Stores real output matrix.

P
: Number of rows in B.

Q
: Number of columns in B.

Discussion
The image is given by the input matrix A. It has M rows and N columns.

$$C_{(m+(P\text{-}1)/2,\,n+(Q\text{-}1)/2)} = \sum_{p=0}^{P-1} \sum_{q=0}^{Q-1} A_{(m+p,n+q)} \cdot B_{(p,q)} \qquad \text{m} = \{0, \text{M-P}\} \text{ and n} = \{0, \text{N-Q}\}$$

B is the filter kernel. It has P rows and Q columns.

Ensure Q >= P for best performance.

The filtered image is placed in the output matrix C. The function pads the perimeter of the output image with a border of (P-1)/2 rows of zeros on the top and bottom and (Q-1)/2 columns of zeros on the left and right.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_imgfirD

Filters an image by performing a two-dimensional convolution with a kernel; double precision.

```
void vDSP_imgfirD (double * A,
vDSP_Length M,
vDSP_Length N,
double * B,
double * C,
vDSP_Length P,
vDSP_Length Q);
```

Parameters

A
A complex vector signal input.

M
Number of rows in input matrix.

N
Number of columns in input matrix.

B
A two-dimensional real matrix containing the filter.

C
Stores real output matrix.

P
Number of rows in B.

Q
Number of columns in B.

Discussion
The image is given by the input matrix A. It has M rows and N columns.

$$C_{(m+(P\text{-}1)/2,\,n+(Q\text{-}1)/2)} = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} A_{(m+p,n+q)} \cdot B_{(p,q)} \qquad \mathrm{m} = \{0, \mathrm{M\text{-}P}\} \text{ and } \mathrm{n} = \{0, \mathrm{N\text{-}Q}\}$$

B is the filter kernel. It has P rows and Q columns. For best performance, ensure Q >= P.

The filtered image is placed in the output matrix C. The functions pad the perimeter of the output image with a border of (P-1)/2 rows of zeros on the top and bottom and (Q-1)/2 columns of zeros on the left and right.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_wiener

Wiener-Levinson general convolution; single precision.

```
void vDSP_wiener (vDSP_Length L,
float * A,
float * C,
float * F,
float * P,
int IFLG,
int * IERR);
```

Parameters

L
: Input filter length

A
: Single-precision real input vector: coefficients

C
: Single-precision real input vector: input coefficients

F
: Single-precision real output vector: filter coefficients

P
: Single-precision real output vector: error prediction operators

IFLG
: Not currently used, pass zero

IERR
: Error flag

Discussion
Performs the operation

$$\text{Find} \quad C_m \quad \text{such that} \quad F_n = \sum_{m=0}^{L-1} A_{n-m} \cdot C_m \qquad n = \{0, L\text{-}1\}$$

solves a set of single-channel normal equations described by:

```
B[n] = C[0] * A[n] + C[1] * A[n-1] +, . . . ,+ C[N-1] * A[n-N+1]
for n = {0, N-1}
```

where matrix `A` contains elements of the symmetric Toeplitz matrix shown below. This function can only be done out of place.

Note that `A`[-n] is considered to be equal to `A`[n].

`vDSP_wiener` solves this set of simultaneous equations using a recursive method described by Levinson. See Robinson, E.A., *Multichannel Time Series Analysis with Digital Computer Programs*. San Francisco: Holden-Day, 1967, pp. 43-46.

```
|A[0]  A[1]  A[2] ... A[N-1] |    |C[0]  |   |B[0]  |
|A[1]  A[0]  A[1] ... A[N-2] |    |C[1]  |   |B[1]  |
|A[2]  A[1]  A[0] ... A[N-3] |  * |C[2]  | = |B[2]  |
| ...   ...   ... ... ...    |    | ...  |   | ...  |
|A[N-1]A[N-2]A[N-3] ... A[0] |    |C[N-1]|   |B[N-1]|
```

Typical methods for solving N equations in N unknowns have execution times proportional to N cubed, and memory requirements proportional to N squared. By taking advantage of duplicate elements, the recursion method executes in a time proportional to N squared and requires memory proportional to N. The Wiener-Levinson algorithm recursively builds a solution by computing the m+1 matrix solution from the m matrix solution.

With successful completion, `vDSP_wiener` returns zero in error flag `IERR`. If `vDSP_wiener` fails, `IERR` indicates in which pass the failure occurred.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_wienerD

Wiener-Levinson general convolution; double precision.

```
void vDSP_wienerD (vDSP_Length L,
double * A,
double * C,
double * F,
double * P,
int IFLG,
int * IERR);
```

Parameters

L
Input filter length

A
Double-precision real input vector: coefficients

C
Double-precision real input vector: input coefficients

F
Double-precision real output vector: filter coefficients

P
Double-precision real output vector: error prediction operators

IFLG
Not currently used, pass zero

IERR
Error flag

Discussion

Performs the operation

$$\text{Find} \quad C_m \quad \text{such that} \quad F_n = \sum_{m=0}^{L-1} A_{n-m} \cdot C_m \qquad n = \{0, L-1\}$$

solves a set of single-channel normal equations described by:

```
B[n] = C[0] * A[n] + C[1] * A[n-1] +, . . . ,+ C[N-1] * A[n-N+1]
for n = {0, N-1}
```

where matrix A contains elements of the symmetric Toeplitz matrix shown below. This function can only be done out of place.

Note that A[-n] is considered to be equal to A[n].

vDSP_wiener solves this set of simultaneous equations using a recursive method described by Levinson. See Robinson, E.A., *Multichannel Time Series Analysis with Digital Computer Programs*. San Francisco: Holden-Day, 1967, pp. 43-46.

```
|A[0]  A[1]  A[2] ... A[N-1] |     |C[0]  |    |B[0]  |
|A[1]  A[0]  A[1] ... A[N-2] |     |C[1]  |    |B[1]  |
|A[2]  A[1]  A[0] ... A[N-3] |   * |C[2]  | =  |B[2]  |
| ...    ...    ... ... ...  |     | ...  |    | ...  |
|A[N-1]A[N-2]A[N-3] ... A[0] |     |C[N-1]|    |B[N-1]|
```

Typical methods for solving N equations in N unknowns have execution times proportional to N cubed, and memory requirements proportional to N squared. By taking advantage of duplicate elements, the recursion method executes in a time proportional to N squared and requires memory proportional to N. The Wiener-Levinson algorithm recursively builds a solution by computing the m+1 matrix solution from the m matrix solution.

With successful completion, vDSP_wiener returns zero in error flag IERR. If vDSP_wiener fails, IERR indicates in which pass the failure occurred.

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_zconv

Performs either correlation or convolution on two complex vectors; single precision.

```
void vDSP_zconv (DSPSplitComplex * signal,
vDSP_Stride signalStride,
DSPSplitComplex * filter,
vDSP_Stride strideFilter,
DSPSplitComplex * result,
vDSP_Stride strideResult,
vDSP_Length lenResult,
vDSP_Length lenFilter);
```

Discussion
A is the input vector, with stride I, and C is the output vector, with stride K and length N.

B is a filter vector, with stride I and length P. If Jis positive,the function performs correlation. If Jis negative, it performs convolution and Bmust point to the last element in the filter vector. The function can run in place, but Ccannot be in place with B.

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad n = \{0, N\text{-}1\}$$

The value of N must be less than or equal to 512.

Criteria to invoke vectorized code:

- Both the real parts and the imaginary parts of vectors A and C must be relatively aligned.
- The values of I and K must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_zconvD

Performs either correlation or convolution on two complex vectors; double precision.

```
void vDSP_zconvD (DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
DSPDoubleSplitComplex * filter,
vDSP_Stride strideFilter,
DSPDoubleSplitComplex * result,
vDSP_Stride strideResult,
vDSP_Length lenResult,
vDSP_Length lenFilter);
```

Discussion
A is the input vector, with stride I, and C is the output vector, with stride K and length N.

B is a filter vector, with stride I and length P. If J is positive, the function performs correlation. If J is negative, it performs convolution and B must point to the last element in the filter vector. The function can run in place, but C cannot be in place with B.

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad \text{n = \{0, N-1\}}$$

The value of N must be less than or equal to 512.

Criteria to invoke vectorized code:

No Altivec support for double precision. On a PowerPC processor, this function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_zrdesamp

Complex/real downsample with anti-aliasing; single precision.

```
void vDSP_zrdesamp (DSPSplitComplex * A,
vDSP_Stride I,
float * B,
DSPSplitComplex * C,
vDSP_Length N,
vDSP_Length M);
```

Parameters

A
: Single-precision complex input vector.

I
: Complex decimation factor.

B
: Filter coefficient vector.

C
: Single-precision complex output vector.

N
: Length of output vector.

M
: Length of real filter vector.

Discussion
Performs finite impulse response (FIR) filtering at selected positions of input vector A.

$$C_m = \sum_{p=0}^{P-1} A_{(mi+p)} \cdot B_p , \qquad \text{(m = \{0, N-1\})}$$

Length of A must be at least (N+M-1)*i. This function can run in place, but C cannot be in place with B.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP_zrdesampD

Complex/real downsample with anti-aliasing; double precision.

```
void vDSP_zrdesampD (DSPDoubleSplitComplex * A,
vDSP_Stride I,
double * B,
DSPDoubleSplitComplex * C,
vDSP_Length N,
vDSP_Length M);
```

Parameters

A
: Double-precision complex input vector.

I
: Complex decimation factor.

B
: Filter coefficient vector.

C
: Double-precision complex output vector.

N
: Length of output vector.

M
: Length of real filter vector.

Discussion
Performs finite impulse response (FIR) filtering at selected positions of input vector A.

$$C_m = \sum_{p=0}^{P-1} A_{(mi+p)} \cdot B_p , \qquad (\mathrm{m} = \{0, \mathrm{N}\text{-}1\})$$

Length of A must be at least (N+M-1)*i. This function can run in place, but C cannot be in place with B.

Availability
Available in Mac OS X v10.4 and later.

Declared In
vDSP.h

vDSP One-Dimensional Fast Fourier Transforms Reference

Framework:	Accelerate/vecLib
Declared in	vDSP.h

Overview

This document describes the C API for performing one-dimensional Fast Fourier Transforms on an input signal. It also describes the `FFTSetup` structure which you pass as a handle to the FFT functions.

Functions by Task

Creating and Freeing FFT Setups

`vDSP_create_fftsetup` (page 260)
Builds a data structure that contains precalculated data for use by single-precision FFT functions.

`vDSP_create_fftsetupD` (page 261)
Builds a data structure that contains precalculated data for use by double-precision FFT functions.

`vDSP_destroy_fftsetup` (page 262)
Frees an existing single-precision FFT data structure.

`vDSP_destroy_fftsetupD` (page 262)
Frees an existing double-precision FFT data structure.

Computing In-Place Complex FFTs

`vDSP_fft_zip` (page 288)
Computes an in-place single-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zip` (page 267)
Performs the same operation as `VDSP_fft_zip` on multiple signals with a single call.

`vDSP_fft_zipD` (page 289)
Computes an in-place double-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zipD` (page 268)
: Performs the same operation as `VDSP_fft_zipD` on multiple signals with a single call.

`vDSP_fft_zipt` (page 290)
: Computes an in-place single-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse). A buffer is used for intermediate results.

`vDSP_fftm_zipt` (page 269)
: Performs the same operation as `VDSP_fft_zipt` on multiple signals with a single call.

`vDSP_fft_ziptD` (page 291)
: Computes an in-place double-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse). A buffer is used for intermediate results.

`vDSP_fftm_ziptD` (page 271)
: Performs the same operation as VDSP_fft_ziptD on multiple signals with a single call.

Computing Out-of-Place Complex FFTs

`vDSP_fft_zop` (page 292)
: Computes an out-of-place single-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fft_zopD` (page 293)
: Computes an out-of-place double-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zop` (page 272)
: Performs the same operation as `VDSP_fft_zop` on multiple signals with a single call.

`vDSP_fftm_zopD` (page 273)
: Performs the same operation as `VDSP_fft_zopD` on multiple signals with a single call.

`vDSP_fft_zopt` (page 294)
: Computes an out-of-place single-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fft_zoptD` (page 296)
: Computes an out-of-place double-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zopt` (page 275)
: Performs the same operation as `VDSP_fft_zopt` on multiple signals with a single call.

`vDSP_fftm_zoptD` (page 276)
: Performs the same operation as `VDSP_fft_zoptD` on multiple signals with a single call.

`vDSP_fft3_zop` (page 263)
: Computes an out-of-place radix-3 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 3 times the power of 2 specified by parameter `log2n`; single precision.

`vDSP_fft3_zopD` (page 264)
Computes an out-of-place radix-3 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 3 times the power of 2 specified by parameter `log2n`; double precision.

`vDSP_fft5_zop` (page 265)
Computes an out-of-place radix-5 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 5 times the power of 2 specified by parameter `log2n`; single precision.

`vDSP_fft5_zopD` (page 266)
Computes an out-of-place radix-5 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 5 times the power of 2 specified by parameter `log2n`; double precision.

Computing In-Place Real FFTs

`vDSP_fft_zrip` (page 297)
Computes an in-place single-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zrip` (page 278)
Performs the same operation as `VDSP_fft_zrip` on multiple signals with a single call.

`vDSP_fft_zripD` (page 298)
Computes an in-place double-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zripD` (page 279)
Performs the same operation as `VDSP_fft_zripD` on multiple signals with a single call.

`vDSP_fft_zript` (page 299)
Computes an in-place single-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zript` (page 280)
Performs the same operation as `VDSP_fft_zript` on multiple signals with a single call.

`vDSP_fft_zriptD` (page 300)
Computes an in-place double-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zriptD` (page 281)
Performs the same operation as `VDSP_fft_zriptD` on multiple signals with a single call.

Computing Out-of-Place Real FFTs

`vDSP_fft_zrop` (page 301)
Computes an out-of-place single-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

`vDSP_fftm_zrop` (page 282)
Performs the same operation as `VDSP_fft_zrop` on multiple signals with a single call.

Functions

vDSP_create_fftsetup

Builds a data structure that contains precalculated data for use by single-precision FFT functions.

```
FFTSetup vDSP_create_fftsetup (vDSP_Length log2n, FFTRadix radix);
```

Parameters

log2n
: A base 2 exponent that represents the number of divisions of the complex unit circle and thus specifies the largest power of two that can be processed by a subsequent frequency-domain function. Parameter `log2n` must equal or exceed the largest power of 2 that any subsequent function processes using the weights array.

radix
: Specifies radix options. Radix 2, radix 3, and radix 5 functions are supported.

Return Value

Returns an `FFTSetup` structure for use with one-dimensional FFT functions. Returns 0 on error.

Discussion

This function allocates memory for an `FFTSetup` data structure needed by FFT functions, initializes that memory, and then returns it. Once prepared, the setup structure can be used repeatedly by FFT functions (which read the data in the structure and do not alter it) for any (power of two) length up to that specified when you created the structure.

If an application performs FFTs with diverse lengths, the calls with different lengths can share a single setup structure (created for the longest length), and this saves space over having multiple structures. However, in some cases, notably single-precision FFTs on 32-bit PowerPC, an FFT routine may perform faster if it is passed a setup structure that was created specifically for the length of the transform.

Parameter log2n is a base-two exponent and specifies that the largest transform length that can processed using the resulting setup structure is `2**log2n` (or `3*2**log2n` or `5*2**log2n` if the appropriate flags are passed, as discussed below). That is, the `log2n` parameter must equal or exceed the value passed to any subsequent FFT routine using the setup structure returned by this routine.

Parameter radix specifies radix options. Its value may be the bitwise OR of any combination of `FFT_RADIX2`, `FFT_RADIX3`, or `FFT_RADIX5`. The resulting setup structure may be used with any of the routines for which the respective flag was used. (The radix-3 and radix-5 FFT routines have "fft3" and "fft5" in their names. The radix-2 FFT routines have plain "fft" in their names.)

If zero is returned, the routine failed to allocate storage.

The setup structure is deallocated by calling `vDSP_destroy_fftsetup`.

Use `vDSP_create_fftsetup` during initialization. It is relatively slow compared to the routines that actually perform FFTs. Never use it in a part of an application that needs to be high performance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_create_fftsetupD

Builds a data structure that contains precalculated data for use by double-precision FFT functions.

```
FFTSetupD vDSP_create_fftsetupD (vDSP_Length log2n, FFTRadix radix);
```

Parameters

log2n

A base 2 exponent that represents the number of divisions of the complex unit circle and thus specifies the largest power of two that can be processed by a subsequent frequency-domain function. Parameter `log2n` must equal or exceed the largest power of 2 that any subsequent function processes using the weights array.

radix

Specifies radix options. Radix 2, radix 3, and radix 5 functions are supported.

Return Value
Returns an `FFTSetupD` structure for use with FFT functions, or 0 if there was an error.

Discussion
This function allocates memory for an `FFTSetup` data structure needed by FFT functions, initializes that memory, and then returns it. Once prepared, the setup structure can be used repeatedly by FFT functions (which read the data in the structure and do not alter it) for any (power of two) length up to that specified when you created the structure.

If an application performs FFTs with diverse lengths, the calls with different lengths can share a single setup structure (created for the longest length), and this saves space over having multiple structures. However, in some cases, notably single-precision FFTs on 32-bit PowerPC, an FFT routine may perform faster if it is passed a setup structure that was created specifically for the length of the transform.

Parameter log2n is a base-two exponent and specifies that the largest transform length that can processed using the resulting setup structure is `2**log2n` (or `3*2**log2n` or `5*2**log2n` if the appropriate flags are passed, as discussed below). That is, the `log2n` parameter must equal or exceed the value passed to any subsequent FFT routine using the setup structure returned by this routine.

Parameter radix specifies radix options. Its value may be the bitwise OR of any combination of `FFT_RADIX2`, `FFT_RADIX3`, or `FFT_RADIX5`. The resulting setup structure may be used with any of the routines for which the respective flag was used. (The radix-3 and radix-5 FFT routines have "fft3" and "fft5" in their names. The radix-2 FFT routines have plain "fft" in their names.)

If zero is returned, the routine failed to allocate storage.

The setup structure is deallocated by calling `vDSP_destroy_fftsetupD`.

Use `vDSP_create_fftsetupD` during initialization. It is relatively slow compared to the routines that actually perform FFTs. Never use it in a part of an application that needs to be high performance.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_destroy_fftsetup

Frees an existing single-precision FFT data structure.

```
void vDSP_destroy_fftsetup (FFTSetup setup);
```

Parameters

`setup`
: Identifies the weights array, and must point to a data structure previously created by `vDSP_create_fftsetup`

Discussion
`vDSP_destroy_fftsetup` frees an existing weights array. Any memory allocated for the array is released. After the `vDSP_destroy_fftsetup` function returns, the structure should not be used in any other functions.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_destroy_fftsetupD

Frees an existing double-precision FFT data structure.

```
void vDSP_destroy_fftsetupD (FFTSetupD setup);
```

Parameters

setup

Identifies the weights array, and must point to a data structure previously created by `vDSP_create_fftsetupD`.

Discussion

`vDSP_destroy_fftsetupD` frees an existing weights array. Any memory allocated for the array is released. After the `vDSP_destroy_fftsetupD` function returns, the structure should not be used in any other functions.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft3_zop

Computes an out-of-place radix-3 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 3 times the power of 2 specified by parameter `log2n`; single precision.

```
void vDSP_fft3_zop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
DSPSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Length log2n,
FFTDirection flag);
```

Parameters

setup

Use `vDSP_create_fftsetup`, to initialize this function. `FFT_RADIX3` must be specified in the call to `vDSP_create_fftsetup`. `setup` is preserved for reuse.

signal

A complex vector signal input.

signalStride

Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

result

The complex vector signal output.

resultStride

Specifies an address stride for the result. The value of `resultStride` should be 1 for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. `log2n` must be between 3 and 15, inclusive.

flag

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
This performs the operation

Where $N = 3(2^m)$ for Radix-3 functions, and $N = 5(2^m)$ for Radix-5 functions

If F = 1 $C_m = \mathrm{RDFT}(A_m) \bullet 2$ If F = -1 $C_m = \mathrm{IDFT}(A_m) \bullet N$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft3_zopD

Computes an out-of-place radix-3 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 3 times the power of 2 specified by parameter `log2n`; double precision.

```
void vDSP_fft3_zopD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride K,
DSPDoubleSplitComplex * result,
vDSP_Stride L,
vDSP_Length log2n,
FFTDirection flag);
```

Parameters

setup
: Use `vDSP_create_fftsetupD`, to initialize this function. `FFT_RADIX3` must be specified in the call to `vDSP_create_fftsetupD`. `setup` is preserved for reuse.

signal
: A complex vector input.

K
: Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `K`; to process every other element, specify 2. The value of `K` should be 1 for best performance.

result
: The complex vector result.

L
: Specifies an address stride for the result. The value of `L` should be 1 for best performance.

log2n
: The base 2 exponent of the number of elements to process in a single input signal. `log2n` must be between 3 and 15, inclusive.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $\quad C_m = \text{RDFT}(A_m) \cdot 2 \qquad$ If F = -1 $\quad C_m = \text{IDFT}(A_m) \cdot N \qquad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft5_zop

Computes an out-of-place radix-5 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 5 times the power of 2 specified by parameter `log2n`; single precision.

```
void vDSP_fft5_zop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
DSPSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Length log2n,
FFTDirection flag);
```

Parameters

setup
: Use `vDSP_create_fftsetup`, to initialize this function. `FFT_RADIX5` must be specified in the call to `vDSP_create_fftsetup`. `setup` is preserved for reuse.

signal
: A complex vector signal input.

signalStride

Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

result

The complex vector signal output.

resultStride

Specifies an address stride for the result. The value of `resultStride` should be 1 for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. `log2n` must be between 3 and 15, inclusive.

flag

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $\quad C_m = \mathrm{RDFT}(A_m) \bullet 2 \qquad$ If F = -1 $\quad C_m = \mathrm{IDFT}\ (A_m) \bullet N \qquad$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft5_zopD

Computes an out-of-place radix-5 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 5 times the power of 2 specified by parameter `log2n`; double precision.

```
void vDSP_fft5_zopD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride K,
DSPDoubleSplitComplex * result,
vDSP_Stride L,
vDSP_Length log2n,
FFTDirection flag);
```

Parameters

setup
: Use `vDSP_create_fftsetupD`, to initialize this function. `FFT_RADIX5` must be specified in the call to `vDSP_create_fftsetupD`. `setup` is preserved for reuse.

signal
: A complex vector input.

K
: Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `K`; to process every other element, specify 2.

result
: The complex vector result.

L
: Specifies an address stride for the result.

log2n
: The base 2 exponent of the number of elements to process in a single input signal.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $\quad C_m = \mathrm{RDFT}(A_m) \bullet 2 \qquad$ If F = -1 $\quad C_m = \mathrm{IDFT}(A_m) \bullet N \qquad$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zip

Performs the same operation as `VDSP_fft_zip` on multiple signals with a single call.

```
void vDSP_fftm_zip (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` (page 260). The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector that stores the input and output signal.

signalStride
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

log2n
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`.

numFFT
: The number of signals.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This function allows you to perform Discrete Fourier Transforms on multiple input signals using a single call. They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride.

The functions compute in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zipD

Performs the same operation as `VDSP_fft_zipD` on multiple signals with a single call.

```
void vDSP_fftm_zipD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector that stores the input and output signal.

signalStride
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

log2n
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT
: The number of signals.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This function allows you to perform Discrete Fourier Transforms on multiple signals at once, using a single call. It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride.

The function computes in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zipt

Performs the same operation as `VDSP_fft_zipt` on multiple signals with a single call.

```
void vDSP_fftm_zipt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal

A complex vector that stores the input and output signal.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

temp

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT

The number of different input signals.

flag

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fftm_ziptD

Performs the same operation as VDSP_fft_ziptD on multiple signals with a single call.

```
void vDSP_fftm_ziptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPDoubleSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector that stores the input and output signal.

signalStride
Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride
The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

temp
A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n
The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT
The number of different input signals.

flag
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zop

Performs the same operation as `VDSP_fft_zop` on multiple signals with a single call.

```
void vDSP_fftm_zop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup`. The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal

A complex vector that stores the input signal.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result

The complex vector signal output.

resultStride
Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride
The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n
The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`.

numFFT
The number of input signals.

flag
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
The function allows you to perform Discrete Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, `signal`, and single output vector, `result`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fftm_zopD

Performs the same operation as `VDSP_fft_zopD` on multiple signals with a single call.

```
void vDSP_fftm_zopD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPDoubleSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal

A complex vector signal input.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result

The complex vector signal output.

resultStride

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT

The number of different input signals.

flag

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fftm_zopt

Performs the same operation as `VDSP_fft_zopt` on multiple signals with a single call.

```
void vDSP_fftm_zopt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
DSPSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector signal input.

signalStride
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result
: The complex vector signal output.

resultStride
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

temp
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT
: The number of different input signals.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zoptD

Performs the same operation as `VDSP_fft_zoptD` on multiple signals with a single call.

```
void vDSP_fftm_zoptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPDoubleSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
DSPDoubleSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal

A complex vector signal input.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result

The complex vector signal output.

resultStride

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2.

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

temp

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT

The number of different input signals.

flag

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zrip

Performs the same operation as `VDSP_fft_zrip` on multiple signals with a single call.

```
void vDSP_fftm_zrip (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` (page 260). The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal

A complex vector signal input.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter `signalStride`; to process every other element, specify 2.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`.

numFFT

The number of input signals.

flag
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
The function allows you to perform Discrete Fourier Transforms on multiple signals using a single call. They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride.

The functions compute in-place real Discrete Fourier Transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fftm_zripD

Performs the same operation as `VDSP_fft_zripD` on multiple signals with a single call.

```
void vDSP_fftm_zripD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD` (page 261). The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector signal input.

signalStride
Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter `signalStride`; to process every other element, specify 2.

fftStride
The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

log2n
The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`.

numFFT
The number of input signals.

flag
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride.

The functions compute in-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zript

Performs the same operation as `VDSP_fft_zript` on multiple signals with a single call.

```
void vDSP_fftm_zript (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` (page 260). The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector signal input.

signalStride
Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter `signalStride`; to process every other element, specify 2.

fftStride
The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

temp

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`.

numFFT

The number of different input signals.

flag

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride.

The functions compute in-place real Discrete Fourier Transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zriptD

Performs the same operation as `VDSP_fft_zriptD` on multiple signals with a single call.

```
void vDSP_fftm_zriptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPDoubleSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD` (page 261). The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector signal input.

signalStride
Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2.

fftStride
The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

temp
A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n
The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`.

numFFT
The number of input signals.

flag
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride.

The functions compute in-place real Discrete Fourier Transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fftm_zrop

Performs the same operation as `VDSP_fft_zrop` on multiple signals with a single call.

```
void vDSP_fftm_zrop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector signal input.

signalStride
: Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result
: The complex vector signal output.

resultStride
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2.

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

numFFT
: The number of input signals.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This function allows you to perform Discrete Fourier Transforms on multiple input signals using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, `signal`, and a single output vector, `result`.

The functions compute out-of-place real Discrete Fourier Transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fftm_zropD

Performs the same operation as `VDSP_fft_zropD` on multiple signals with a single call.

```
void vDSP_fftm_zropD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPDoubleSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector signal input.

signalStride
: Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result
: The complex vector signal output.

resultStride
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

numFFT
The number of input signals.

flag
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This function allows you to perform Discrete Fourier Transforms on multiple input signals using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, the parameter `signal`.

The functions compute out-of-place real Discrete Fourier Transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zropt

Performs the same operation as `VDSP_fft_zropt` on multiple signals with a single call.

```
void vDSP_fftm_zropt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
DSPSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector signal input.

signalStride
: Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result
: The complex vector signal output.

resultStride
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

temp
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

numFFT
: The number of input signals.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, the parameter `signal`.

The functions compute out-of-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetup" (page 260), "vDSP_destroy_fftsetup" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fftm_zroptD

Performs the same operation as `VDSP_fft_zroptD` on multiple signals with a single call.

```
void vDSP_fftm_zroptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
vDSP_Stride fftStride,
DSPDoubleSplitComplex * result,
vDSP_Stride resultStride,
vDSP_Stride rfftStride,
DSPDoubleSplitComplex * temp,
vDSP_Length log2n,
vDSP_Length numFFT,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector signal input.

signalStride
: Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

result
: The complex vector signal output.

resultStride
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

temp
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `temp.realp` and `temp.imagp` should be 32-byte aligned for best performance.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

numFFT
: The number of different input signals.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, the parameter `signal`.

The functions compute out-of-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

See also functions "vDSP_create_fftsetupD" (page 261), "vDSP_destroy_fftsetupD" (page 262), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft_zip

Computes an in-place single-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zip (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride stride,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` .The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector that stores the input and output signal.

stride
: Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `stride`; to process every other element, specify 2.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
This performs the operation

$$\text{If F} = 1 \qquad C_m = \text{FDFT}(C_m) \qquad \text{If F} = -1 \qquad C_m = \text{IDFT}\,(C_m) \bullet N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zipD

Computes an in-place double-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zipD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride stride,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD` (page 261). The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector that stores the input and output signal.

stride
: Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `stride`; to process every other element, specify 2. The value of `stride` should be 1 for best performance.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
This performs the operation

If F = 1 $\quad C_m = \text{FDFT}(C_m) \quad$ If F = -1 $\quad C_m = \text{IDFT}(C_m) \bullet N \quad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N} \sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zipt

Computes an in-place single-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft_zipt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride stride,
DSPSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector that stores the input and output signal.

stride
: Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `stride`; to process every other element, specify 2.

bufferTemp
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `bufferTemp.realp` and `bufferTemp.imagp` should be 32-byte aligned for best performance.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(C_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}(C_m) \cdot N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_ziptD

Computes an in-place double-precision complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft_ziptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride stride,
DSPDoubleSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector that stores the input and output signal.

stride
Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `stride`; to process every other element, specify 2.

bufferTemp
A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `bufferTemp.realp` and `bufferTemp.imagp` should be 32-byte aligned for best performance.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

$$\text{If F} = 1 \quad C_m = \text{FDFT}(C_m) \qquad \text{If F} = -1 \quad C_m = \text{IDFT}(C_m) \bullet N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft_zop

Computes an out-of-place single-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
DSPSplitComplex * result,
vDSP_Stride strideResult,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal

A complex vector that stores the input and output signal.

signalStride

Specifies an address stride through input vector signal. Parameter `strideResult` specifies an address stride through output vector result. Thus, to process every element, specify a signalStride of 1; to process every other element, specify 2.

result

The complex vector signal output.

strideResult
Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(A_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}(A_m) \bullet N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft_zopD

Computes an out-of-place double-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zopD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
DSPDoubleSplitComplex * result,
vDSP_Stride strideResult,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector that stores the input signal.

signalStride
Specifies an address stride through input vector signal. Parameter `strideResult` specifies an address stride through output vector result. Thus, to process every element, specify a `signalStride` of 1; to process every other element, specify 2. The values of `signalStride` and `strideResult` should be 1 for best performance.

result
The complex vector signal output.

strideResult
Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `strideResult` should be 1 for best performance.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(A_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}(A_m) \cdot N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft_zopt

Computes an out-of-place single-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zopt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
DSPSplitComplex * result,
vDSP_Stride strideResult,
DSPSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector that stores the input and output signal.

signalStride
: Specifies an address stride through input vector signal. Parameter `strideResult` specifies an address stride through output vector result. Thus, to process every element, specify a signalStride of 1; to process every other element, specify 2. The values of signalStride and `strideResult` should be 1 for best performance.

result
: The complex vector signal output.

strideResult
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2.

bufferTemp
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer.`realp` and tempBuffer.`imagp` should be 32-byte aligned for best performance.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

$$\text{If F = 1} \quad C_m = \mathrm{FDFT}(A_m) \qquad \text{If F = -1} \quad C_m = \mathrm{IDFT}\,(A_m) \bullet N \qquad \mathrm{m} = \{0, \mathrm{N}\text{-}1\}$$

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zoptD

Computes an out-of-place double-precision complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zoptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
DSPDoubleSplitComplex * result,
vDSP_Stride strideResult,
DSPDoubleSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
A complex vector signal input.

signalStride
Specifies an address stride through input vector signal. Parameter `strideResult` specifies an address stride through output vector result. Thus, to process every element, specify a signalStride of 1; to process every other element, specify 2. The values of signalStride and `strideResult` should be 1 for best performance.

result
The complex vector signal output.

strideResult
Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2.

bufferTemp
A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer.`realp` and tempBuffer.`imagp` should be 32-byte aligned for best performance.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(A_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}(A_m) \bullet N \qquad \text{m = \{0, N-1\}}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N} \sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zrip

Computes an in-place single-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zrip (FFTSetup setup,
DSPSplitComplex * C,
vDSP_Stride K,
vDSP_Length log2n,
FFTDirection F);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` (page 260). The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

C
: A complex input/output vector.

K
: Specifies an address stride through the input/output vector. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

F
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
Forward transforms read real input and write packed complex output. You can find more details on the packing format in vDSP Library. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $C_m = \mathrm{RDFT}(C_m) \cdot 2$ If F = -1 $C_m = \mathrm{IDFT}\ (C_m) \cdot N$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N} \sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zripD

Computes an in-place double-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zripD (FFTSetupD setup,
DSPDoubleSplitComplex * C,
vDSP_Stride K,
vDSP_Length log2n,
FFTDirection F);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetupD` (page 261). The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

C
A complex vector input.

K
Specifies an address stride through the input/output vector . To process every element of the vector, specify 1 for parameter `stride`; to process every other element, specify 2.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

F
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $\quad C_m = \mathrm{RDFT}(C_m) \bullet 2 \qquad$ If F = -1 $\quad C_m = \mathrm{IDFT}\ (C_m) \bullet N \qquad$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N} \sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zript

Computes an in-place single-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zript (FFTSetup setup,
DSPSplitComplex * C,
vDSP_Stride K,
DSPSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection F);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` (page 260). The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

C
: A complex vector input.

K
: Specifies an address stride through the input/output vector . To process every element of the vector, specify 1 for parameter signalStride; to process every other element, specify 2.

bufferTemp
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer.`realp` and tempBuffer.`imagp` should be 32-byte aligned for best performance.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

F

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

$$\text{If } F = 1 \quad C_m = \text{RDFT}(C_m) \cdot 2 \qquad \text{If } F = -1 \quad C_m = \text{IDFT}\ (C_m) \cdot N \qquad m = \{0, N\text{-}1\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft_zriptD

Computes an in-place double-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zriptD (FFTSetupD setup,
DSPDoubleSplitComplex * C,
vDSP_Stride K,
DSPDoubleSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection F);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetupD` (page 261). The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

C

A complex vector input.

K

Specifies an address stride through the input/output vector . To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2.

bufferTemp
A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer.`realp` and tempBuffer.`imagp` should be 32-byte aligned for best performance.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

F
A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

$$\text{If F = 1} \qquad C_m = \text{RDFT}(C_m) \bullet 2 \qquad \text{If F = -1} \qquad C_m = \text{IDFT}\ (C_m) \bullet N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft_zrop

Computes an out-of-place single-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zrop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
DSPSplitComplex * result,
vDSP_Stride strideResult,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector signal input.

signalStride
: Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

result
: The complex vector signal output.

strideResult
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

$$\text{If F} = 1 \quad C_m = \text{RDFT}(A_m) \bullet 2 \qquad \text{If F} = -1 \quad C_m = \text{IDFT}\ (A_m) \bullet N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output (see vDSP Library for details on the packing format). Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zropD

Computes an out-of-place double-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zropD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
DSPDoubleSplitComplex * result,
vDSP_Stride strideResult,
vDSP_Length log2n,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector signal input.

signalStride
: Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

result
: The complex vector signal output.

strideResult
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `strideResult` should be 1 for best performance.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

flag
: A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion
This performs the operation

If F = 1 $\quad C_m = \mathrm{RDFT}(A_m) \bullet 2$ $\quad$ If F = -1 $\quad C_m = \mathrm{IDFT}\ (A_m) \bullet N$ $\quad$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements. Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft_zropt

Computes an out-of-place single-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zropt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStride,
DSPSplitComplex * result,
vDSP_Stride strideResult,
DSPSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal
: A complex vector signal input.

signalStride
: Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

result
: The complex vector signal output.

strideResult
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `strideResult` should be 1 for best performance.

bufferTemp
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer`.realp` and tempBuffer`.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

direction

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

$$\text{If F} = 1 \quad C_m = \mathrm{RDFT}(A_m) \bullet 2 \qquad \text{If F} = -1 \quad C_m = \mathrm{IDFT}\,(A_m) \bullet N \qquad \mathrm{m} = \{0, \mathrm{N}\text{-}1\}$$

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements. Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions "vDSP_create_fftsetup" (page 260) and "vDSP_destroy_fftsetup" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft_zroptD

Computes an out-of-place double-precision real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void vDSP_fft_zroptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStride,
DSPDoubleSplitComplex * result,
vDSP_Stride strideResult,
DSPDoubleSplitComplex * bufferTemp,
vDSP_Length log2n,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of this transform function.

signal

A complex vector signal input.

signalStride

Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

result

The complex vector signal output.

strideResult

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `strideResult` should be 1 for best performance.

bufferTemp

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer.`realp` and tempBuffer.`imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`.

flag

A forward/inverse directional flag, which must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Discussion

This performs the operation

$$\text{If F = 1} \quad C_m = \text{RDFT}(A_m) \cdot 2 \qquad \text{If F = -1} \quad C_m = \text{IDFT}(A_m) \cdot N \qquad \text{m = \{0, N-1\}}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements. Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions "vDSP_create_fftsetupD" (page 261) and "vDSP_destroy_fftsetupD" (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP Two-Dimensional Fast Fourier Transforms Reference

Framework: Accelerate/vecLib

Declared in vDSP.h

Overview

This document describes the C API for performing two-dimensional Fast Fourier Transforms on an input signal. It also describes the `FFTSetup` structure which you pass as a handle to the FFT functions.

Functions by Task

Computing In-Place Complex FFTs

`vDSP_fft2d_zip` (page 309)
Computes an in-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zipD` (page 310)
Computes an in-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zipt` (page 312)
Computes an in-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

`vDSP_fft2d_ziptD` (page 313)
Computes an in-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

Computing Out-of-Place Complex FFTs

`vDSP_fft2d_zop` (page 315)
Computes an out-of-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zopD` (page 316)
Computes an out-of-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zopt` (page 317)
: Computes an out-of-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

`vDSP_fft2d_zoptD` (page 319)
: Computes an out-of-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

Computing In-Place Real FFTs

`vDSP_fft2d_zrip` (page 320)
: Computes an in-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zripD` (page 322)
: Computes an in-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zript` (page 324)
: Computes an in-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

`vDSP_fft2d_zriptD` (page 325)
: Computes an in-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

Computing Out-of-Place Real FFTs

`vDSP_fft2d_zrop` (page 327)
: Computes an out-of-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zropD` (page 329)
: Computes an out-of-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

`vDSP_fft2d_zropt` (page 330)
: Computes an out-of-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

`vDSP_fft2d_zroptD` (page 332)
: Computes an out-of-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

Functions

vDSP_fft2d_zip

Computes an in-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zip (FFTSetup setup,
DSPSplitComplex * ioData,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

ioData

A complex vector input.

strideInRow

Specifies a stride across each row of the matrix `signal`. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

If not 0, parameter `strideInCol` represents the distance between each row of the matrix. If parameter `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_{nm} = \mathrm{FDFT2D}(C_{nm}) \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$

If F = -1 $\quad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \bullet MN \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zipD

Computes an in-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zipD (FFTSetupD setup,
DSPDoubleSplitComplex * ioData,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection direction);
```

Parameters

`setup`
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

`ioData`
: A complex vector input.

`strideInRow`
: Specifies a stride across each row of the matrix `signal`. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

If not 0, parameter `strideInCol` represents the distance between each row of the matrix. If parameter `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `direction`.

Discussion

This performs the operation

$$\text{If F = 1} \qquad C_{nm} = \text{FDFT2D}(C_{nm}) \qquad \text{n = \{0, N-1\} and m = \{0, M-1\}}$$

$$\text{If F = -1} \qquad C_{nm} = \text{IDFT2D}(C_{nm}) \bullet MN \qquad \text{n = \{0, N-1\} and m = \{0, M-1\}}$$

$$\text{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\text{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft2d_zipt

Computes an in-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_zipt (FFTSetup setup,
DSPSplitComplex * ioData,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
DSPSplitComplex * bufferTemp,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

ioData

A complex vector input.

strideInRow

Specifies a stride across each row of the matrix `signal`. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

If not 0, parameter `strideInCol` represents the distance between each row of the matrix. If parameter `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

bufferTemp

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_{nm} = \mathrm{FDFT2D}(C_{nm}) \quad$ n = {0, N-1} and m = {0, M-1}

If F = -1 $\quad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \cdot MN \quad$ n = {0, N-1} and m = {0, M-1}

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft2d_ziptD

Computes an in-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_ziptD (FFTSetupD setup,
DSPDoubleSplitComplex * ioData,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
DSPDoubleSplitComplex * bufferTemp,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection direction);
```

Parameters

`setup`

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

`ioData`

A complex vector input.

`strideInRow`

Specifies a stride across each row of the matrix `signal`. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol
Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

If not 0, parameter `strideInCol` represents the distance between each row of the matrix. If parameter `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

bufferTemp
A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol
The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction
A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `direction`.

Discussion
This performs the operation

$$\text{If } F = 1 \qquad C_{nm} = \mathrm{FDFT2D}(C_{nm}) \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

$$\text{If } F = -1 \qquad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \cdot MN \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

See also functions "`vDSP_create_fftsetup`", "`vDSP_create_fftsetupD`", "`vDSP_destroy_fftsetup`", and "`vDSP_destroy_fftsetupD`".

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zop

Computes an out-of-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStrideInRow,
vDSP_Stride signalStrideInCol,
DSPSplitComplex * result,
vDSP_Stride strideResultInRow,
vDSP_Stride strideResultInCol,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

signal
: A complex vector signal input.

signalStrideInRow
: Specifies a stride across each row of matrix `a`. Specifying 1 for `signalStrideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalStrideInCol
: If not 0, this parameter represents the distance between each row of the input /output matrix.

result
: The complex vector signal output.

strideResultInRow
: Specifies a row stride for output matrix `result` in the same way that `signalStrideInRow` specifies a stride for input the input /output matrix.

strideResultInCol
: Specifies a column stride for output matrix `result` in the same way that `signalStrideInCol` specifies a stride for input the input /output matrix.

log2nInCol
: The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
: The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

flag
: A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

: Results are undefined for other values of `flag`.

Discussion

This performs the operation

If F = 1 $\quad C_m = \text{FDFT}(A_m)$ $\quad$ If F = -1 $\quad C_m = \text{IDFT}\,(A_m) \bullet N$ $\quad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zopD

Computes an out-of-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zopD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStrideInRow,
vDSP_Stride signalStrideInCol,
DSPDoubleSplitComplex * result,
vDSP_Stride strideResultInRow,
vDSP_Stride strideResultInCol,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

signal
: A complex vector signal input.

signalStrideInRow
: Specifies a stride across each row of matrix `a`. Specifying 1 for `signalStrideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalStrideInCol
: If not 0, this parameter represents the distance between each row of the input /output matrix.

result
: The complex vector signal output.

strideResultInRow
: Specifies a row stride for output matrix `result` in the same way that `signalStrideInRow` specifies a stride for input the input /output matrix.

strideResultInCol
Specifies a column stride for output matrix `result` in the same way that `signalStrideInCol` specifies a stride for input the input /output matrix.

log2nInCol
The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

flag
A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `flag`.

Discussion
This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(A_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}\ (A_m) \bullet N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zopt

Computes an out-of-place single-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_zopt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStrideInRow,
vDSP_Stride signalStrideInCol,
DSPSplitComplex * result,
vDSP_Stride strideResultInRow,
vDSP_Stride strideResultInCol,
DSPSplitComplex * bufferTemp,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

signal

A complex vector signal input.

signalStrideInRow

Specifies a stride across each row of matrix `a`. Specifying 1 for `signalStrideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalStrideInCol

If not 0, this parameter represents the distance between each row of the input /output matrix. If parameter signalStrideInCol is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

result

The complex vector signal output.

strideResultInRow

Specifies a row stride for output matrix `result` in the same way that `signalStrideInRow` specifies a stride for input the input /output matrix.

strideResultInCol

Specifies a column stride for output matrix `result` in the same way that `signalStrideInCol` specifies a stride for input the input /output matrix.

bufferTemp

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KiB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

flag

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `flag`.

Discussion
This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(A_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}(A_m) \bullet N \qquad \text{m = \{0, N-1\}}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zoptD

Computes an out-of-place double-precision complex discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_zoptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride signalStrideInRow,
vDSP_Stride signalStrideInCol,
DSPDoubleSplitComplex * result,
vDSP_Stride strideResultInRow,
vDSP_Stride strideResultInCol,
DSPDoubleSplitComplex * bufferTemp,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

`setup`
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

`signal`
: A complex vector signal input.

`signalStrideInRow`
: Specifies a stride across each row of matrix `a`. Specifying 1 for `signalStrideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

`signalStrideInCol`
: If not 0, this parameter represents the distance between each row of the input /output matrix. If parameter signalStrideInCol is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

result
: The complex vector signal output.

strideResultInRow
: Specifies a row stride for output matrix `result` in the same way that `signalStrideInRow` specifies a stride for input the input /output matrix.

strideResultInCol
: Specifies a column stride for output matrix `result` in the same way that `signalStrideInCol` specifies a stride for input the input /output matrix.

bufferTemp
: A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol
: The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
: The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

flag
: A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

  Results are undefined for other values of `flag`.

Discussion

This performs the operation

$$\text{If } F = 1 \quad C_m = \text{FDFT}(A_m) \qquad \text{If } F = -1 \quad C_m = \text{IDFT}(A_m) \bullet N \qquad m = \{0, N\text{-}1\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft2d_zrip

Computes an in-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zrip (FFTSetup setup,
DSPSplitComplex * ioData,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and `log2nInRow` of the transform function.

ioData
: A complex vector input.

strideInRow
: Specifies a stride across each row of the input matrix signal. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol
: Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

  If not 0, `strideInCol` represents the distance between each row of the matrix. If `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

log2nInCol
: The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
: The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction
: A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

  Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $\quad C_{nm} = \text{FDFT2D}(C_{nm}) \cdot 2 \qquad$ n = {0, N-1} and m = {0, M-1}

If F = -1 $\quad C_{nm} = \text{IDFT2D}(C_{nm}) \cdot MN \qquad$ n = {0, N-1} and m = {0, M-1}

$$\text{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\text{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zripD

Computes an in-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zripD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

`setup`
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and `log2nInRow` of the transform function.

`signal`
: A complex vector signal input.

`strideInRow`
: Specifies a stride across each row of the input matrix signal. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol
Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

If not 0, `strideInCol` represents the distance between each row of the matrix. If `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

log2nInCol
The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

flag
A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `flag`.

Discussion
Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $\quad C_{nm} = \mathrm{FDFT2D}(C_{nm}) \bullet 2 \qquad$ n = {0, N-1} and m = {0, M-1}

If F = -1 $\quad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \bullet MN \qquad$ n = {0, N-1} and m = {0, M-1}

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

No Altivec/SSE support for double precision. The function always invokes scalar code.

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zript

Computes an in-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_zript (FFTSetup setup,
DSPSplitComplex * ioData,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
DSPSplitComplex * bufferTemp,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and `log2nInRow` of the transform function.

ioData

A complex vector input.

strideInRow

Specifies a stride across each row of the input matrix signal. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

If not 0, `strideInCol` represents the distance between each row of the matrix. If `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

bufferTemp

A temporary matrix used for storing interim results. The size of temporary memory required is discussed below.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

$$\text{If } F = 1 \qquad C_{nm} = \text{FDFT2D}(C_{nm}) \cdot 2 \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

$$\text{If } F = -1 \qquad C_{nm} = \text{IDFT2D}(C_{nm}) \cdot MN \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

$$\text{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\text{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

The space needed in `bufferTemp` is at most max(9*nr, nc/2) elements in each of `realp` and `imagp`. Here is an example of how to allocate the space:

```
int nr, nc, tempSize;
nr = 1<<log2InRow;
nc = 1<<log2InCol;
tempSize = max(9*nr, nc/2);
bufferTemp.realp = ( float* ) malloc (tempSize * sizeOf ( float ) );
bufferTemp.imagp = ( float* ) malloc (tempSize * sizeOf ( float ) );
```

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft2d_zriptD

Computes an in-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_zriptD (FFTSetupD setup,
DSPDoubleSplitComplex * signal,
vDSP_Stride strideInRow,
vDSP_Stride strideInCol,
DSPDoubleSplitComplex * bufferTemp,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and `log2nInRow` of the transform function.

signal

A complex vector signal input.

strideInRow

Specifies a stride across each row of the input matrix signal. Specifying 1 for `strideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

strideInCol

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `strideInCol` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `strideInRow` is 1 and `strideInCol` is 0, every element of the input /output matrix is processed. If `strideInRow` is 2 and `strideInCol` is 0, every other element of each row is processed.

If not 0, `strideInCol` represents the distance between each row of the matrix. If `strideInCol` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

bufferTemp

A temporary matrix used for storing interim results. The size of temporary memory required is discussed below.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

flag

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `flag`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $\quad C_{nm} = \text{FDFT2D}(C_{nm}) \bullet 2 \qquad$ n = {0, N-1} and m = {0, M-1}

If F = -1 $\quad C_{nm} = \text{IDFT2D}(C_{nm}) \bullet MN \qquad$ n = {0, N-1} and m = {0, M-1}

$$\text{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\text{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

The space needed in `bufferTemp` is at most max(9*nr, nc/2) elements in each of `realp` and `imagp`. Here is an example of how to allocate the space:

```
int nr, nc, tempSize;
nr = 1<<log2InRow;
nc = 1<<log2InCol;
tempSize = max(9*nr, nc/2);
bufferTemp.realp = ( float* ) malloc (tempSize * sizeOf ( float ) );
bufferTemp.imagp = ( float* ) malloc (tempSize * sizeOf ( float ) );
```

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zrop

Computes an out-of-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zrop (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStrideInRow,
vDSP_Stride signalStrideInCol,
DSPSplitComplex * result,
vDSP_Stride strideResultInRow,
vDSP_Stride strideResultInCol,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

signal

A complex vector signal input.

signalStrideInRow

Specifies a stride across each row of matrix signal. Specifying 1 for `signalStrideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalStrideInCol

If not 0, represents the distance between each row of the input /output matrix. If parameter signalStrideInCol is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

result

The complex vector signal output.

strideResultInRow

Specifies a row stride for output matrix `c` in the same way that `signalStrideInRow` specifies strides for input the matrix.

strideResultInCol

Specifies a column stride for output matrix `c` in the same way that `signalStrideInCol` specify strides for input the matrix.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

flag

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `flag`.

Discussion
Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_fft2d_zropD

Computes an out-of-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void vDSP_fft2d_zropD (FFTSetupD setup,
DSPDoubleSplitComplex * ioData,
vDSP_Stride Kr,
vDSP_Stride Kc,
DSPDoubleSplitComplex * ioData2,
vDSP_Stride Ir,
vDSP_Stride Ic,
vDSP_Length log2nc,
vDSP_Length log2nr,
FFTDirection flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

ioData
: A complex vector input.

Kr
: Specifies a stride across each row of matrix signal. Specifying 1 for `Kr` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

Kc
: If not 0, represents the distance between each row of the input /output matrix. If parameter `Kc` is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

ioData2
: The complex vector result.

Ir
Specifies a row stride for output matrix `ioData2` in the same way that `Kr` specifies strides for input the matrix.

Ic
Specifies a column stride for output matrix `ioData2` in the same way that `Kc` specify strides for input matrix `ioData`.

log2nc
The base 2 exponent of the number of columns to process for each row. `log2nc` must be between 3 and 10, inclusive.

log2nr
The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nc` and 6 for `log2nr`. `log2nr` must be between 3 and 10, inclusive.

flag
A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `flag`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_fft2d_zropt

Computes an out-of-place single-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_zropt (FFTSetup setup,
DSPSplitComplex * signal,
vDSP_Stride signalStrideInRow,
vDSP_Stride signalStrideInCol,
DSPSplitComplex * result,
vDSP_Stride strideResultInRow,
vDSP_Stride strideResultInCol,
DSPSplitComplex * bufferTemp,
vDSP_Length log2nInCol,
vDSP_Length log2nInRow,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

signal

A complex vector signal input.

signalStrideInRow

Specifies a stride across each row of matrix `signal`. Specifying 1 for `signalStrideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalStrideInCol

If not 0, represents the distance between each row of matrix `signal`. If parameter `signalStrideInCol` is 1024, for instance, element 512 equates to element (1,0) of matrix `signal`, element 1024 equates to element (2,0), and so forth.

result

The complex vector signal output.

strideResultInRow

Specifies a row stride for output matrix `result` in the same way that `signalStrideInRow` specifies strides for input matrix `result`.

strideResultInCol

Specifies a column stride for output matrix `c` in the same way that `signalStrideInCol` specify strides for input matrix `result`.

bufferTemp

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) can be calculated using the algorithm shown below.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

flag

A forward/inverse directional flag, and must specify FFT_FORWARD for a forward transform or FFT_INVERSE for an inverse transform.

Results are undefined for other values of flag.

Discussion

Here is the bufferTemp size algorithm:

```
int nr, nc, tempSize;
nr = 1<<log2InRow;
nc = 1<<log2InCol;
if ( ( (log2InCol-1) < 3 ) || ( log2InRow > 9)
{
tempSize = 9 * nr;
}
else
{
tempSize = 17 * nr
}
bufferTemp.realp = (float*) malloc (tempSize * sizeOf (float));
bufferTemp.imagp = (float*) malloc (tempSize * sizeOf (float));
```

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions vDSP_create_fftsetup (page 260), vDSP_create_fftsetupD (page 261), vDSP_destroy_fftsetup (page 262), and vDSP_destroy_fftsetupD (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

vDSP.h

vDSP_fft2d_zroptD

Computes an out-of-place double-precision real discrete FFT, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse). A buffer is used for intermediate results.

```
void vDSP_fft2d_zroptD (FFTSetupD setup,
DSPDoubleSplitComplex * ioData,
vDSP_Stride Kr,
vDSP_Stride Kc,
DSPDoubleSplitComplex * ioData2,
vDSP_Stride Ir,
vDSP_Stride Ic,
DSPDoubleSplitComplex * temp,
vDSP_Length log2nc,
vDSP_Length log2nr,
FFTDirection flag);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

ioData

A complex vector input.

Kr

Specifies a stride across each row of matrix signal. Specifying 1 for `signalStrideInRow` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

Kc

If not 0, represents the distance between each row of the input /output matrix. If parameter signalStrideInCol is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

ioData2

The complex vector result.

Ir

Specifies a row stride for output matrix `ioData2` in the same way that `Kr` specifies strides for input the matrix.

Ic

Specifies a column stride for output matrix `ioData2` in the same way that `Kc` specify strides for input matrix `ioData`.

temp

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nc

The base 2 exponent of the number of columns to process for each row. `log2nc` must be between 3 and 10, inclusive.

log2nr

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nc` and 6 for `log2nr`. `log2nr` must be between 3 and 10, inclusive.

flag

A forward/inverse directional flag, and must specify `FFT_FORWARD` for a forward transform or `FFT_INVERSE` for an inverse transform.

Results are undefined for other values of `flag`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements.

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

See also functions `vDSP_create_fftsetup` (page 260), `vDSP_create_fftsetupD` (page 261), `vDSP_destroy_fftsetup` (page 262), and `vDSP_destroy_fftsetupD` (page 262).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP Complex Vector Conversion Reference

Framework: Accelerate/vecLib

Declared in vDSP.h

Overview

Describes the C API for the vecLib functions that convert complex vectors between interleaved and split forms.

Functions

vDSP_ctoz

Copies the contents of an interleaved complex vector `C` to a split complex vector `Z`; single precision.

```
void vDSP_ctoz (const DSPComplex C[],
vDSP_Stride strideC,
DSPSplitComplex * Z,
vDSP_Stride strideZ,
vDSP_Length size);
```

Discussion
Performs the operation

$$A_{nI} = Re(C_{nK}) \quad ; \quad A_{nK+1} = Im(C_{n;K}) \qquad n = \{0, N\text{-}1\}$$

`strideC` is an address stride through C. `strideZ` is an address stride through Z. The value of `strideC` must be a multiple of 2.

For best performance, `C.realp`, `C.imagp`, `Z.realp`, and `Z.imagp` should be 16-byte aligned.

See also functions"vDSP_ztoc" (page 336) and "vDSP_ztocD" (page 337).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_ctozD

Copies the contents of an interleaved complex vector C to a split complex vector Z; double precision.

```
void vDSP_ctozD (const DSPDoubleComplex C[],
vDSP_Stride strideC,
DSPDoubleSplitComplex * Z,
vDSP_Stride strideZ,
vDSP_Length size);
```

Discussion

This performs the operation

$$A_{nI} = Re(C_{nK}) \quad ; \quad A_{nK+1} = Im(C_{n;K}) \qquad n = \{0, N\text{-}1\}$$

`strideC` is an address stride through C. `strideZ` is an address stride through Z. The value of `strideC` must be a multiple of 2.

For best performance, `C.realp`, `C.imagp`, `Z.realp`, and `Z.imagp` should be 16-byte aligned.

See also functions "vDSP_ztoc" (page 336) and "vDSP_ztocD" (page 337).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`vDSP.h`

vDSP_ztoc

Copies the contents of a split complex vector A to an interleaved complex vector C; single precision.

```
void vDSP_ztoc (const DSPSplitComplex * Z,
vDSP_Stride strideZ,
DSPComplex C[],
vDSP_Stride strideC,
vDSP_Length size);
```

Discussion

This peforms the operation

$$C_{nK} = Re(A_{nI})$$

$$C_{nK+1} = Im(A_{nI}) \qquad n = \{0, N\text{-}1\}$$

`strideC` is an address stride through `C`. `strideZ` is an address stride through `Z`.

For best performance, `C->realp`, `C->imagp`, `A->realp`, and `A->imagp` should be 16-byte aligned.

See also "vDSP_ctoz" (page 335) and "vDSP_ctozD" (page 336).

Availability

Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

vDSP_ztocD

Copies the contents of a split complex vector `A` to an interleaved complex vector `C`; double precision.

```
void vDSP_ztocD (const DSPDoubleSplitComplex * Z,
vDSP_Stride strideZ,
DSPDoubleComplex C[],
vDSP_Stride strideC,
vDSP_Length size);
```

Discussion
This peforms the operation

$$C_{nK} = Re(A_{nI})$$

$$C_{nK+1} = Im(A_{nI}) \qquad \text{n} = \{0, \text{N-1}\}$$

`strideC` is an address stride through `C`. `strideZ` is an address stride through `Z`.

For best performance, `C->realp`, `C->imagp`, `A->realp`, and `A->imagp` should be 16-byte aligned.

See also "vDSP_ctoz" (page 335) and "vDSP_ctozD" (page 336).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`vDSP.h`

Document Revision History

This table describes the changes to *vDSP Reference Collection*.

Date	Notes
2008-10-15	Fixed several formulaic inaccuracies. Improved discussion on the proper usage of the vDSP_create_fftsetup functions.

Index

V