vDSP Correlation, Convolution, and Filtering Reference

Performance > Carbon

2009-01-06

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

Contents

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 7) 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

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

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

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

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`).

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

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`).

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

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

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

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

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

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

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

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.

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

`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.

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

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

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

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

Document Revision History

This table describes the changes to *vDSP Correlation, Convolution, and Filtering Reference*.

Date	Notes
2009-01-06	Corrected inaccuracies in documenting function parameters.
2008-11-19	Blackman window functions represented in pseudocode.
2007-06-15	New document that describes the C API for the digital signal processing functionality of the vecLib framework.

Document Revision History

Index

V