

---

# vDSP Correlation, Convolution, and Filtering Reference

[Performance > Carbon](#)



2009-01-06



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

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

The Apple logo is a trademark of Apple Inc.

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

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

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

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

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

PowerPC and the PowerPC logo are trademarks of International Business Machines Corporation, used under license therefrom.

Simultaneously published in the United States and Canada.

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

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

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

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

# Contents

---

## **vDSP Correlation, Convolution, and Filtering Reference 5**

---

Overview	5
Functions by Task	5
Correlation and Convolution	5
Windowing and Filtering	6
Functions	6
vDSP_blkman_window	6
vDSP_blkman_windowD	7
vDSP_conv	7
vDSP_convD	8
vDSP_desamp	9
vDSP_desampD	10
vDSP_f3x3	11
vDSP_f3x3D	11
vDSP_f5x5	12
vDSP_f5x5D	13
vDSP_hamm_window	13
vDSP_hamm_windowD	14
vDSP_hann_window	14
vDSP_hann_windowD	15
vDSP_imgfir	16
vDSP_imgfirD	17
vDSP_wiener	18
vDSP_wienerD	19
vDSP_zconv	20
vDSP_zconvD	21
vDSP_zrdesamp	22
vDSP_zrdesampD	23

---

## **Document Revision History 25**

---

## **Index 27**

---



# vDSP Correlation, Convolution, and Filtering Reference

---

<b>Framework:</b>	Accelerate/vecLib
<b>Declared in</b>	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

[vDSP\\_conv](#) (page 7)

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

[vDSP\\_convD](#) (page 8)

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

[vDSP\\_zconv](#) (page 20)

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

[vDSP\\_zconvD](#) (page 21)

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

[vDSP\\_wiener](#) (page 18)

Wiener-Levinson general convolution; single precision.

[vDSP\\_wienerD](#) (page 19)

Wiener-Levinson general convolution; double precision.

[vDSP\\_desamp](#) (page 9)

Convolution with decimation; single precision.

[vDSP\\_desampD](#) (page 10)

Convolution with decimation; double precision.

[vDSP\\_zrdesamp](#) (page 22)

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

[vDSP\\_zrdesampD](#) (page 23)

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

## Windowing and Filtering

[vDSP\\_blkman\\_window](#) (page 6)

Creates a single-precision Blackman window.

[vDSP\\_blkman\\_windowD](#) (page 7)

Creates a double-precision Blackman window.

[vDSP\\_hamm\\_window](#) (page 13)

Creates a single-precision Hamming window.

[vDSP\\_hamm\\_windowD](#) (page 14)

Creates a double-precision Hamming window.

[vDSP\\_hann\\_window](#) (page 14)

Creates a single-precision Hanning window.

[vDSP\\_hann\\_windowD](#) (page 15)

Creates a double-precision Hanning window.

[vDSP\\_f3x3](#) (page 11)

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.

[vDSP\\_f3x3D](#) (page 11)

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.

[vDSP\\_f5x5](#) (page 12)

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.

[vDSP\\_f5x5D](#) (page 13)

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.

[vDSP\\_imgfir](#) (page 16)

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

[vDSP\\_imgfirD](#) (page 17)

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

## 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

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

If `filterStride` is positive, `vDSP_conv` 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:

- 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

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

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

<i>B</i>	Single-precision input filter coefficients
<i>C</i>	Single-precision real output vector
<i>N</i>	Output count
<i>M</i>	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  $\geq (N-1)*I + (\text{nearest multiple of } 4 \geq M)$ .

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

**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**

<i>A</i>	Double-precision real input vector, 8-byte aligned; length of <i>A</i> $\geq 12$
<i>I</i>	Desampling factor
<i>B</i>	Double-precision input filter coefficients
<i>C</i>	Double-precision real output vector
<i>N</i>	Output count
<i>M</i>	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  $\geq (N-1) \cdot I + (\text{nearest multiple of 4} \geq M)$ .

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

**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

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

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.

**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

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

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

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

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

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

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.

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**

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

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**

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

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

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

`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

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

`vDSP_hann_windowD` 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.

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

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

Ensure  $Q \geq P$  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.

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

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

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

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.

$$\begin{array}{c|c|c} |A[0] & A[1] & A[2] & \dots & A[N-1] & | & |C[0] & | & |B[0] & | \\ |A[1] & A[0] & A[1] & \dots & A[N-2] & | & |C[1] & | & |B[1] & | \\ |A[2] & A[1] & A[0] & \dots & A[N-3] & | & |C[2] & | & |B[2] & | \\ | \dots & \dots & \dots & \dots & \dots & | & | \dots & | & | \dots & | \\ |A[N-1] & A[N-2] & A[N-3] & \dots & A[0] & | & |C[N-1] & | & |B[N-1] & | \end{array} * =$$

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

<code>L</code>	Input filter length
<code>A</code>	Double-precision real input vector: coefficients
<code>C</code>	Double-precision real input vector: input coefficients
<code>F</code>	Double-precision real output vector: filter coefficients
<code>P</code>	Double-precision real output vector: error prediction operators
<code>IFLG</code>	Not currently used, pass zero
<code>IERR</code>	Error flag

**Discussion**

Performs the operation

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

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

$$B[n] = C[0] * A[n] + C[1] * A[n-1] + \dots + C[N-1] * A[n-N+1]$$

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.

$$\begin{array}{cccc|ccc} |A[0] & A[1] & A[2] & \dots & A[N-1] & | & C[0] & | & B[0] & | \\ |A[1] & A[0] & A[1] & \dots & A[N-2] & | & C[1] & | & B[1] & | \\ |A[2] & A[1] & A[0] & \dots & A[N-3] & | & C[2] & | & B[2] & | \\ | \dots & \dots & \dots & \dots & \dots & | & \dots & | & \dots & | \\ |A[N-1] & A[N-2] & A[N-3] & \dots & A[0] & | & C[N-1] & | & B[N-1] & | \end{array} * =$$

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.

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

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  $B$ .

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

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  $A$ .

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

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*.

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

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





# 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.

## REVISION HISTORY

### Document Revision History

# Index

---

## V

---

[vDSP\\_blkman\\_window function 6](#)  
[vDSP\\_blkman\\_windowD function 7](#)  
[vDSP\\_conv function 7](#)  
[vDSP\\_convD function 8](#)  
[vDSP\\_desamp function 9](#)  
[vDSP\\_desampD function 10](#)  
[vDSP\\_f3x3 function 11](#)  
[vDSP\\_f3x3D function 11](#)  
[vDSP\\_f5x5 function 12](#)  
[vDSP\\_f5x5D function 13](#)  
[vDSP\\_hamm\\_window function 13](#)  
[vDSP\\_hamm\\_windowD function 14](#)  
[vDSP\\_hann\\_window function 14](#)  
[vDSP\\_hann\\_windowD function 15](#)  
[vDSP\\_imgfir function 16](#)  
[vDSP\\_imgfirD function 17](#)  
[vDSP\\_wiener function 18](#)  
[vDSP\\_wienerD function 19](#)  
[vDSP\\_zconv function 20](#)  
[vDSP\\_zconvD function 21](#)  
[vDSP\\_zrdesamp function 22](#)  
[vDSP\\_zrdesampD function 23](#)