Chapter

11

Fourier Series Method of FIR Filter Design

11.1 Basis of the Fourier Series Method

This Fourier series method of FIR filter design is based on the fact that the frequency response of a digital filter is periodic and is therefore representable as a Fourier series. A desired "target" frequency response is selected and expanded as a Fourier series. This expansion is truncated to a finite number of terms that are then used as the filter coefficients or tap weights. The resulting filter has a frequency response that approximates the original desired target response.

Algorithm 11.1 Designing FIR filters via the Fourier series method

Step 1. Specify a desired frequency response $H_d(\lambda)$.

Step 2. Specify the desired number of filter taps N.

Step 3. Compute the filter coefficients $h[n]$ for $n = 0, 1, 2, \ldots, N-1$ using

$$h[n] = \frac{1}{2\pi} \int_{2\pi} H_d(\lambda)[\cos(m\lambda) + j \sin(m\lambda)]\, d\lambda \tag{11.1}$$

where $m = n - (N-1)/2$.

[Simplifications of (11.1) are presented below for the cases in which H_d is the magnitude response of ideal lowpass, highpass, bandpass, or bandstop filters.]

Step 4. Using the techniques presented in Secs. 10.2 and 10.3, compute the actual frequency response of the resulting filter. If the performance is not adequate, change N or $H_d(\lambda)$ and go back to step 3.

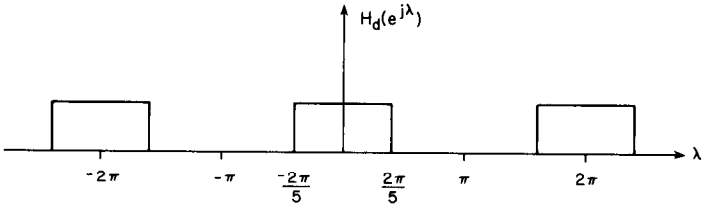

Figure 11.1 Desired frequency response for Example 11.1.

Example 11.1 Use the Fourier series method to design a 21-tap FIR filter that approximates the amplitude response of an ideal lowpass filter with a cutoff frequency of 2 kHz assuming a sampling frequency of 5 kHz.

solution The normalized cutoff is $\lambda = 2\pi/5$. The desired frequency response is depicted in Fig. 11.1. Using Eq. (11.1), we can immediately write

$$h[n] = \frac{1}{2\pi}\int_{-2\pi/5}^{2\pi/5} \cos(m\lambda)\, d\lambda + j\frac{1}{2\pi}\int_{-2\pi/5}^{2\pi/5} \sin(m\lambda)\, d\lambda$$

Since the second integrand is an odd function and the limits of integration are symmetric about zero, the second integral equals zero. Therefore,

$$\begin{aligned} h[n] &= \frac{\sin(m\lambda)}{2m\pi}\bigg|_{\lambda=-2\pi/5}^{2\pi/5} \\ &= \frac{\sin(2m\pi/5)}{m\pi} \end{aligned} \tag{11.2}$$

where $m = n - 10$.

L'Hospital's rule can be used to evaluate (11.2) for the case of $m = 0$ (that is, $n = 10$):

$$\begin{aligned} h[10] &= \frac{(d/dm)\sin(2m\pi/5)}{(d/dm)m\pi}\bigg|_{m=0} \\ &= \frac{(2\pi/5)\cos(2m\pi/5)}{\pi}\bigg|_{m=0} \\ &= \frac{2}{5} = 0.4 \end{aligned}$$

Evaluation of (11.2) for $m \neq 0$ is straightforward. The values of $h[n]$ are listed in Table 11.1, and the corresponding magnitude response is shown in Figs. 11.2 and 11.3. Usually, the pass-band ripples are more pronounced when the vertical axis is in linear units such as numeric magnitude or percentage of peak magnitude as in Fig. 11.2. On the other hand, details of the stop-band response are usually more clearly displayed when the vertical axis is in decibels as in Fig. 11.3.

Properties of the Fourier series method

1. Filters designed using Algorithm 11.1 will exhibit the linear phase property discussed in Sec. 10.3, provided that the target frequency response $H_d(\lambda)$ is either symmetric or antisymmetric.

TABLE 11.1 Impulse Response Coefficients for the 21-tap Lowpass Filter of Example 11.1

$h[0] = h[20] =$	0.000000
$h[1] = h[19] =$	−0.033637
$h[2] = h[18] =$	−0.023387
$h[3] = h[17] =$	0.026728
$h[4] = h[16] =$	0.050455
$h[5] = h[15] =$	0.000000
$h[6] = h[14] =$	−0.075683
$h[7] = h[13] =$	−0.062366
$h[8] = h[12] =$	0.093549
$h[9] = h[11] =$	0.302731
$h[10] =$	0.400000

2. As a consequence of the Gibbs phenomenon, the frequency response of filters designed with Algorithm 11.1 will contain undershoots and overshoots at the band edges as exhibited by the responses shown in Figs. 11.2 and 11.3. As long as the number of filter taps remains finite, these disturbances cannot be eliminated by increasing the number of taps.

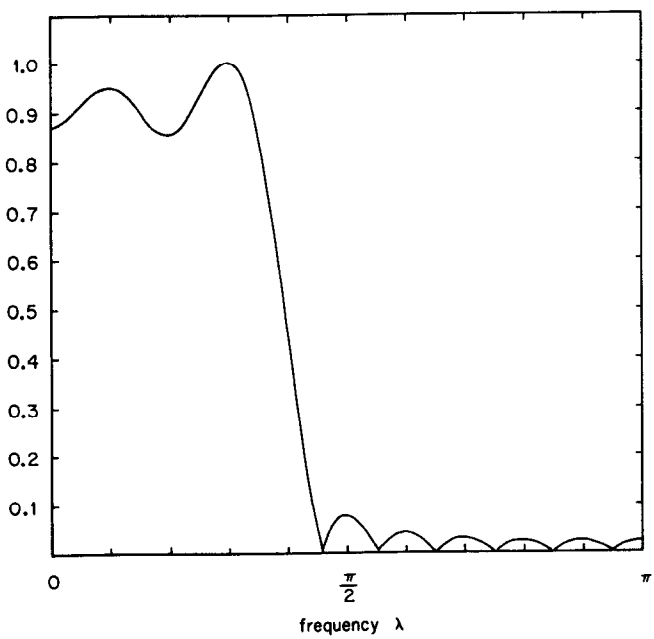

Figure 11.2 Magnitude response (as a percentage of peak) obtained from the 21-tap lowpass filter of Example 11.1.

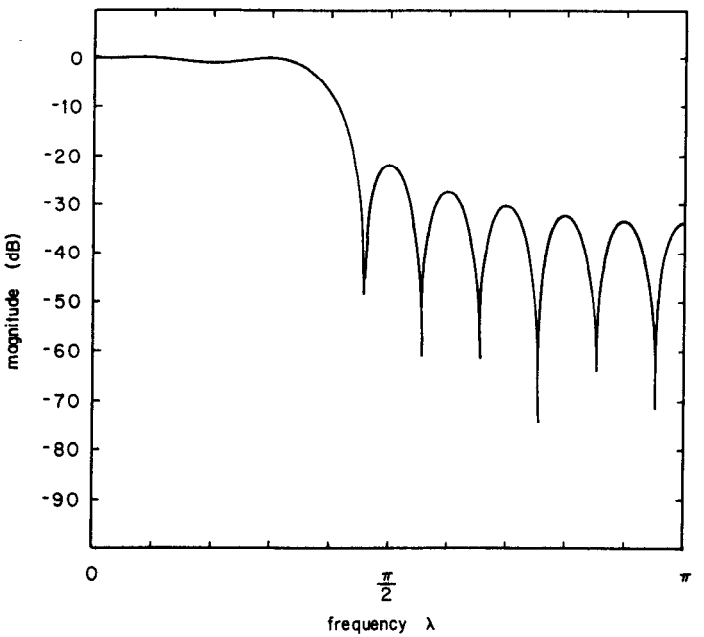

Figure 11.3 Magnitude response (in decibels) obtained from the 21-tap lowpass filter of Example 11.1.

Windowing techniques to reduce the effects of the Gibbs phenomena will be presented later in this chapter.

Result 11.1 FIR approximation for ideal lowpass filter. The impulse response coefficients for an FIR approximation to the ideal lowpass amplitude response shown in Fig. 11.4 are given by

$$h[n] = \frac{\sin(m\lambda_U)}{n\pi} \qquad \begin{matrix} n = 0, 1, \ldots, N-1 \\ m = n - (N-1)/2 \end{matrix}$$

For odd-length filters, the coefficient at $n = (N-1)/2$ is obtained by application of L'Hospital's rule to yield

$$h\left[\frac{N-1}{2}\right] = \frac{\lambda_U}{\pi}$$

The coefficients given by Result 11.1 can be computed using the C function **idealLowpass( )**, which is provided in Listing 11.1.

Result 11.2 FIR approximation for ideal highpass filter. The impulse response coefficients for an FIR approximation to the ideal highpass amplitude response

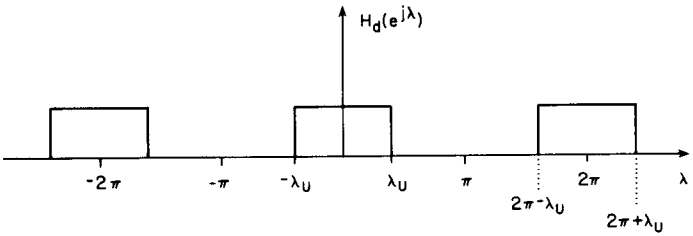

Figure 11.4 Frequency response of ideal lowpass digital filter.

shown in Fig. 11.5 are given by

$$h[n] = \begin{cases} 1 - \dfrac{\lambda_L}{\pi} & m = 0 \\ \dfrac{-\sin(m\lambda_L)}{m\pi} & m \neq 0 \end{cases}$$

where $m = n - (N-1)/2$.

The coefficients given by Result 11.2 can be computed using the C function **idealHighpass( )**, which is provided in Listing 11.2.

Example 11.2 Use Result 11.2 to design a 21-tap FIR filter that approximates the amplitude response of an ideal highpass filter with a normalized cutoff frequency of $\lambda_U = 3\pi/5$.

solution The coefficients $h(n)$ are listed in Table 11.2, and the resulting frequency response is shown in Figs. 11.6 and 11.7.

Result 11.3 FIR approximation for ideal bandpass filter. The impulse response coefficients for an FIR approximation to the ideal bandpass amplitude response shown in Fig. 11.8 are given by

$$h[n] = \begin{cases} \dfrac{\lambda_U - \lambda_L}{\pi} & m = 0 \\ \dfrac{1}{n\pi}[\sin(m\lambda_U) - \sin(m\lambda_L)] & m \neq 0 \end{cases}$$

where $m = n - (N-1)/2$.

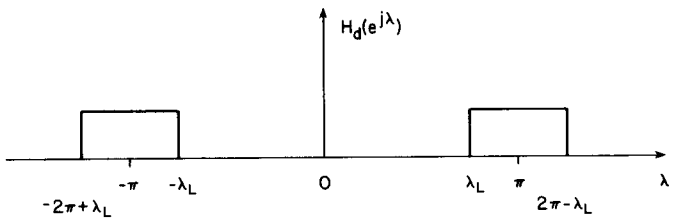

Figure 11.5 Frequency response of ideal highpass digital filter.

TABLE 11.2 Impulse Response Coefficients for the 21-tap Highpass Filter of Example 11.2

$h[0] = h[20] =$	0.000000
$h[1] = h[19] =$	0.033637
$h[2] = h[18] =$	−0.023387
$h[3] = h[17] =$	−0.026728
$h[4] = h[16] =$	0.050455
$h[5] = h[15] =$	0.000000
$h[6] = h[14] =$	−0.075683
$h[7] = h[13] =$	0.062366
$h[8] = h[12] =$	0.093549
$h[9] = h[11] =$	−0.302731
$h[10] =$	0.400000

The coefficients given by Result 11.3 can be computed using the C function **idealBandpass( )**, which is provided in Listing 11.3.

Example 11.3 Use Result 11.3 to design a 21-tap FIR filter that approximates the amplitude response of an ideal bandpass filter with a pass band that extends from $\lambda_L = 2\pi/5$ to $\lambda_U = 3\pi/5$.

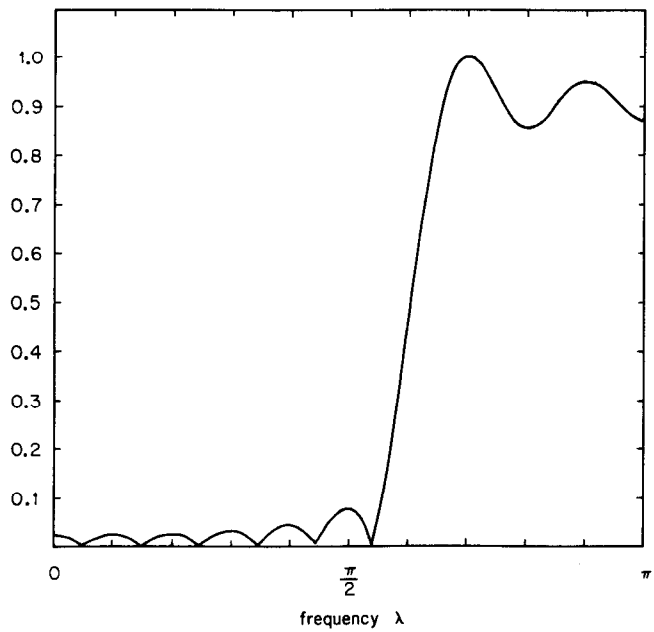

Figure 11.6 Magnitude response (as a percentage of peak) obtained from the 21-tap highpass filter of Example 11.2.

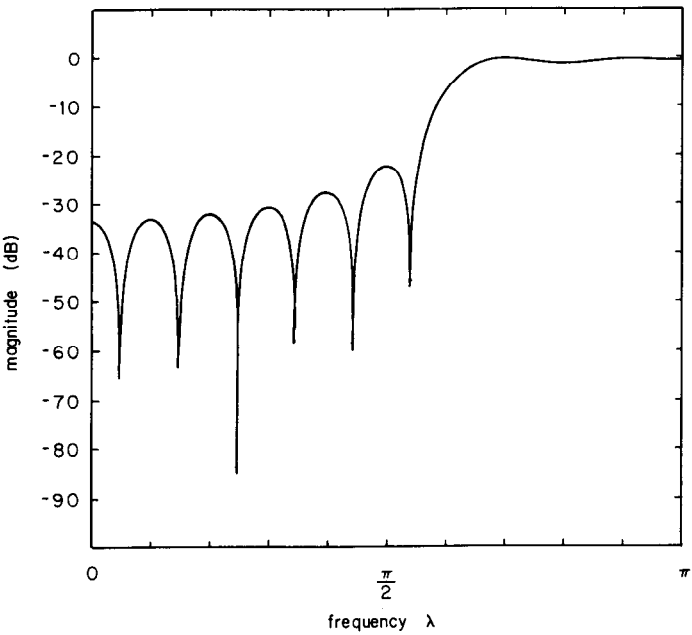

Figure 11.7 Magnitude response (in decibels) obtained from 21-tap highpass filter of Example 11.2.

solution The coefficients $h(n)$ are listed in Table 11.3, and the resulting frequency response is shown in Fig. 11.9.

Result 11.4 FIR approximation for ideal bandstop filter. The impulse response coefficients for an FIR approximation to the ideal bandstop amplitude response shown in Fig. 11.10 are given by

$$h[n] = \begin{cases} 1 + \dfrac{\lambda_L - \lambda_U}{\pi} & m = 0 \\ \dfrac{1}{n\pi}[\sin(m\lambda_L) - \sin(m\lambda_U)] & m \neq 0 \end{cases}$$

where $m = n - (N-1)/2$.

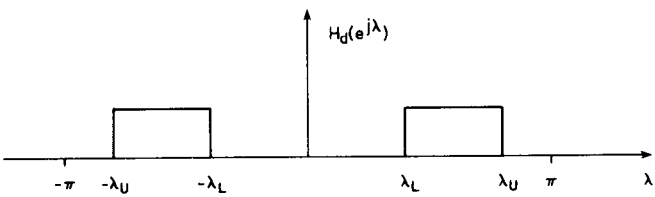

Figure 11.8 Frequency response of ideal bandpass digital filter.

TABLE 11.3 Impulse Response Coefficients for the 21-tap Bandpass Filter of Example 11.3

$h[0] = h[20] =$	0.000000
$h[1] = h[19] =$	0.000000
$h[2] = h[18] =$	0.046774
$h[3] = h[17] =$	0.000000
$h[4] = h[16] =$	−0.100910
$h[5] = h[15] =$	0.000000
$h[6] = h[14] =$	0.151365
$h[7] = h[13] =$	0.000000
$h[8] = h[12] =$	−0.187098
$h[9] = h[11] =$	0.000000
$h[10] =$	0.200000

The coefficients given by Result 11.4 can be computed using the C function **idealBandstop( )**, which is provided in Listing 11.4.

Example 11.4 Use Result 11.4 to design a 31-tap FIR filter that approximates the amplitude response of an ideal bandstop filter with a stop band that extends from $\lambda_L = 2\pi/5$ to $\lambda_U = 3\pi/5$.

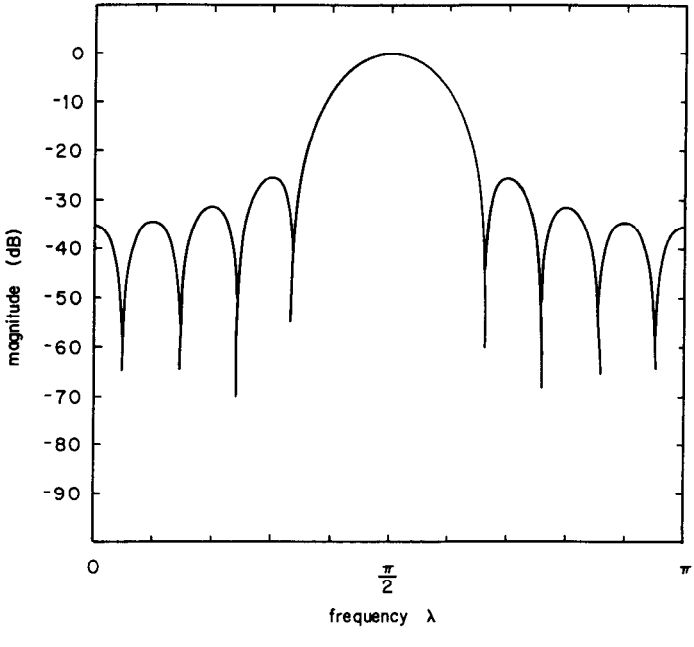

Figure 11.9 Magnitude response (in decibels) obtained from the 21-tap bandpass filter of Example 11.3.

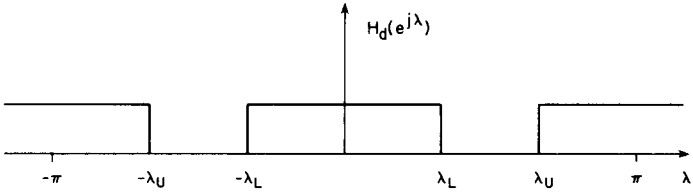

Figure 11.10 Frequency response of ideal bandstop digital filter.

TABLE 11.4 Impulse Response Coefficients for the 31-tap Bandstop Filter of Example 11.4

$h[0] = h[30] =$	0.000000
$h[1] = h[29] =$	−0.043247
$h[2] = h[28] =$	0.000000
$h[3] = h[27] =$	0.031183
$h[4] = h[26] =$	0.000000
$h[5] = h[25] =$	0.000000
$h[6] = h[24] =$	0.000000
$h[7] = h[23] =$	−0.046774
$h[8] = h[22] =$	0.000000
$h[9] = h[21] =$	0.100910
$h[10] = h[20] =$	0.000000
$h[11] = h[19] =$	−0.151365
$h[12] = h[18] =$	0.000000
$h[13] = h[17] =$	0.187098
$h[14] = h[16] =$	0.000000
$h[15] =$	0.800000

solution The coefficients $h(n)$ are listed in Table 11.4, and the resulting frequency response is shown in Figs. 11.11 and 11.12.

11.2 Rectangular Window

As shown in the previous section, filters designed via the Fourier series method will, as a consequence of the Gibbs phenomenon, have frequency responses that contain overshoots and ripple. One way to reduce these effects involves multiplying the filter's impulse response by a *window* that "tapers off" the impulse response instead of abruptly truncating it to a finite number of terms. The basic idea of windowing is very straightforward, and most of the effort in this area is directed toward finding "good" window functions. A discussion of just what constitutes a good window function will be easier if we first develop a windowing viewpoint of truncation.

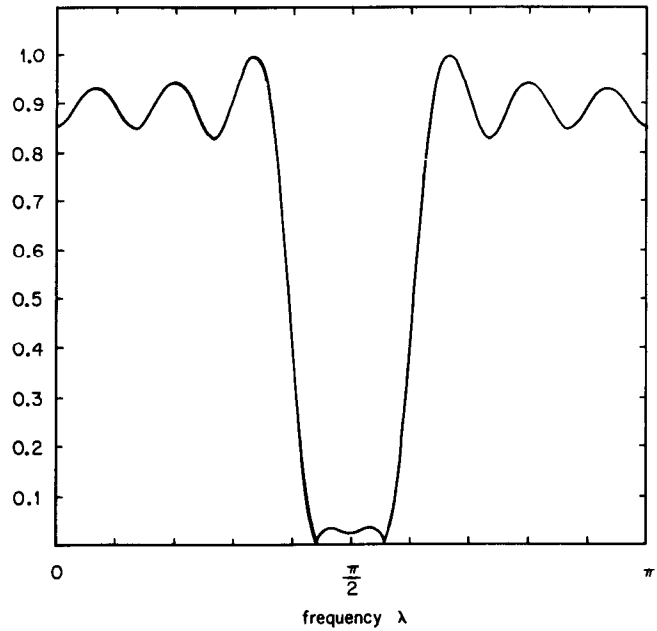

Figure 11.11 Magnitude response (as a percentage of peak) obtained from 31-tap bandstop filter of Example 11.4.

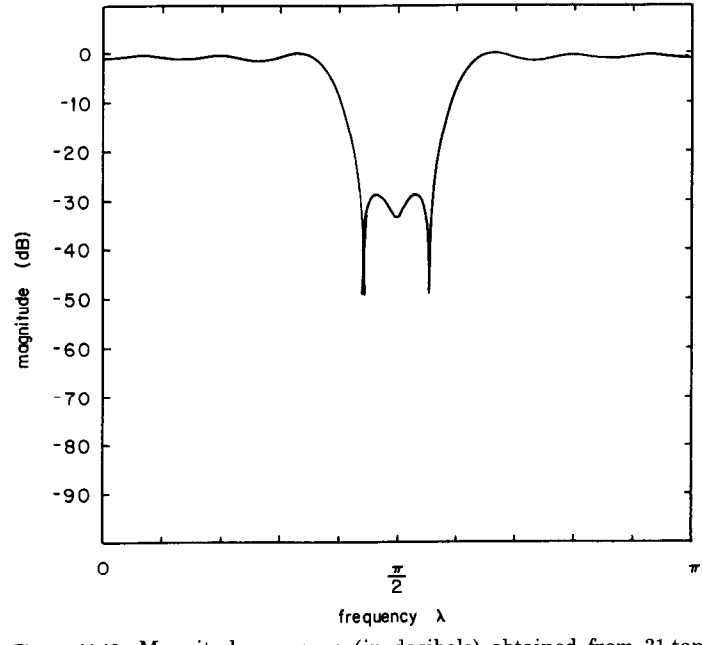

Figure 11.12 Magnitude response (in decibels) obtained from 31-tap bandstop filter of Example 11.4.

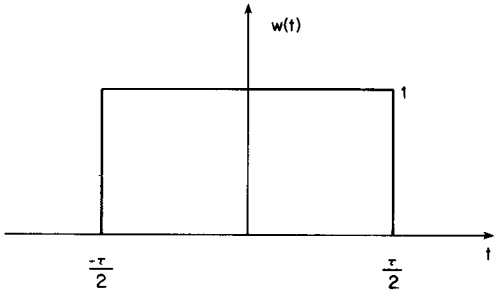

Figure 11.13 Rectangular window.

Truncating a filter's impulse response can be thought of as multiplying the infinite-length impulse response by a rectangular window such as the one shown in Fig. 11.13. This window has a value of unity for all values of t at which the impulse response is to be preserved, and a value of zero for all values of t at which the impulse response is to be eliminated:

$$w(t) = \begin{cases} 1 & |t| < \dfrac{\tau}{2} \\ 0 & \text{otherwise} \end{cases} \tag{11.3}$$

The rectangular window's Fourier transform is given by

$$W(f) = \frac{\tau \sin \pi f\tau}{\pi f\tau} \tag{11.4}$$

The magnitude of (11.4) is plotted in Fig. 11.14. The peaks of the first through ninth sidelobes are attenuated by 13.3, 17.8, 20.8, 23.0, 24.7, 26.2, 27.4, 28.5, and 29.5 dB, respectively. The data for Fig. 11.14 was generated using the C function **contRectangularResponse( )** provided in Listing 11.5.

The rectangular window's response will serve primarily as a benchmark to which the responses of other windows can be compared [*Note*: By omitting further explanation, some texts such as Stanley (1975) imply that Eq. (11.4) also applies to the discrete-time version of the rectangular window. However, as we will discover below, the Fourier transforms of the continuous-time and discrete-time windows differ significantly. A similar situation exists with respect to the triangular window.]

Discrete-time window

Since FIR filter coefficients exist only for integer values of n or discrete values of $t = nT$, it is convenient to work with a window function that is defined in terms of n rather than t. If the function defined by (11.3) is sampled using $N = 2M + 1$ samples with one sample at $t = 0$ and samples at nT for $n = \pm 1, \pm 2, \ldots, \pm M$; the sampled window function becomes

$$w[n] = \begin{cases} 1 & -M \le n \le M \\ 0 & \text{otherwise} \end{cases} \tag{11.5}$$

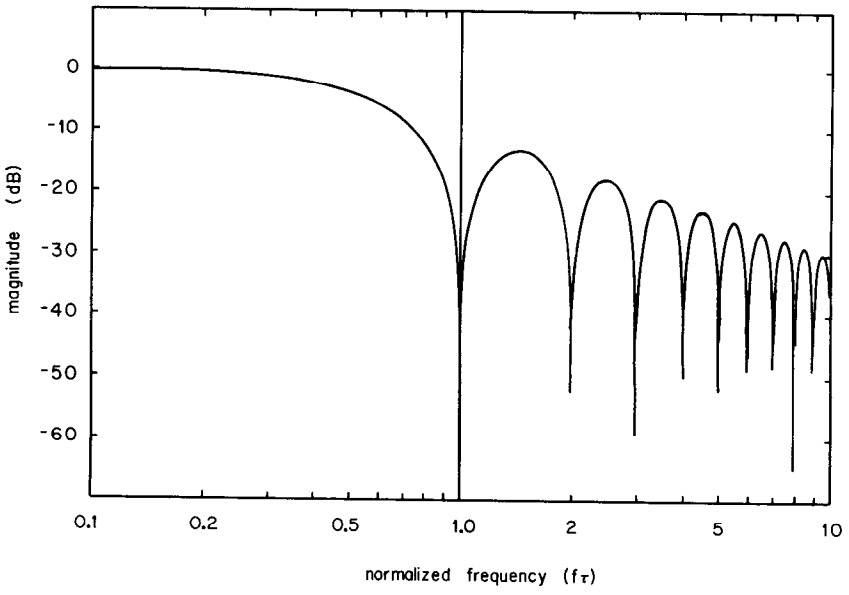

Figure 11.14 Magnitude spectrum for a continuous-time rectangular window.

For an even number of samples, the rectangular window can be defined as either

$$w[n] = 1 \qquad -(M-1) \le n \le M \tag{11.6}$$

or

$$w[n] = 1 \qquad -M \le n \le (M-1) \tag{11.7}$$

The window specified by (11.6) will be centered around a point midway between $n = 0$ and $n = 1$, and the window specified by (11.7) will be centered around a point midway between $n = -1$ and $n = 0$. In many applications (especially in languages like C that use zero-origin indexing), it is convenient to have $w[n]$ defined for $0 \le n \le (N-1)$:

$$w[n] = 1 \qquad 0 \le n \le (N-1) \tag{11.8}$$

In order to emphasize the difference between windows such as (11.5), which are defined over positive and negative frequencies, and windows such as (11.8) which are defined over nonnegative frequencies, digital-signal processing "borrows" terminology from the closely related field of time-series analysis. Using this borrowed terminology, windows such as (11.5) are called *lag windows*, and windows such as (11.8) are called *data windows*. Data windows are also referred to as *tapering windows* and occasionally *tapers* or *faders*. To avoid having to deal with windows centered around $\frac{1}{2}$ or $-\frac{1}{2}$, many authors state that N must be odd for lag windows. However, even-length *data* windows are widely used for leakage reduction in FFT applications.

Frequency windows and spectral windows

The discrete-time Fourier transform (DTFT) of the lag window (11.5) is given by

$$W(f) = \frac{\sin[\pi f(2M+1)]}{\sin(\pi f)} \tag{11.9}$$

The form of (11.9) is closely related to the so-called Dirichlet kernel $D_n(\cdot)$ which is variously defined as

$$D_n(\theta) \triangleq \frac{1}{2\pi} \sum_{k=-n}^{n} \cos k\theta = \frac{\sin\{[n+(1/2)]\theta\}}{\sin(\theta/2)}$$

(Priestley 1981)

$$D_n(x) \triangleq \sum_{k=-n}^{n} \exp(2\pi jkx) = \frac{\sin[(2n+1)\pi x]}{\sin(\pi x)}$$

(Dym and McKean 1972)

$$D_n(x) \triangleq \frac{1}{2} \sum_{k=-n}^{n} \cos(kx) = \frac{\sin\{[n+(1/2)]x\}}{2\sin(x/2)}$$

(Weaver 1989)

The magnitude of (11.9) is plotted in Fig. 11.15 for $N = 11$ and Fig. 11.16 for $N = 21$. As indicated by these two cases, when the number of points in the

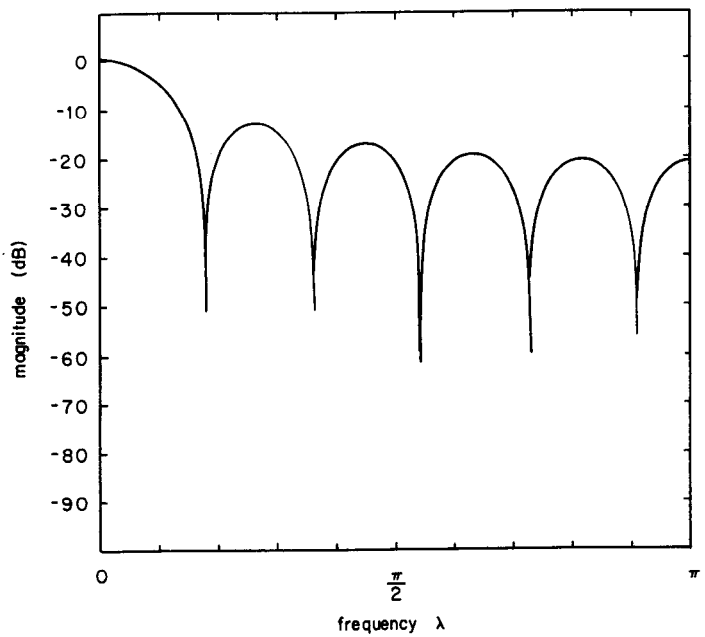

Figure 11.15 Magnitude of the DTFT for an 11-point rectangular window.

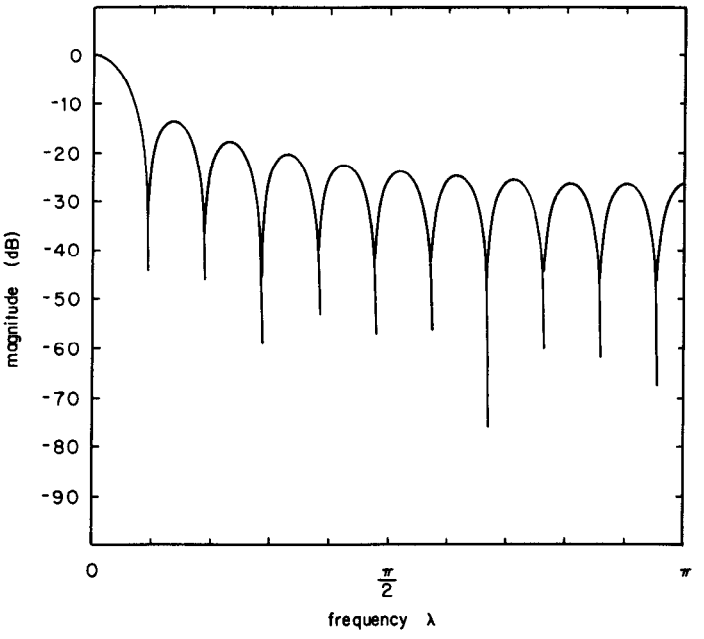

Figure 11.16 Magnitude of the DTFT for a 21-point rectangular window.

window increases, the width of the DTFT sidelobes decreases. The sidelobes in Fig. 11.15 are attenuated by 13.0, 17.1, 19.3, 20.5, and 20.8 dB; and the sidelobes in Fig. 11.16 are attenuated by 13.2, 17.6, 20.4, 22.3, 23.7, 24.8, 25.5, 26.1, and 26.3 dB. The data for these plots were generated using the C function **discRectangularResponse( )** provided in Listing 11.6.

The DTFT of the data window (11.8) is given by

$$W(f) = \exp[-j\pi f(N-1)]\,\frac{\sin(N\pi f)}{\sin(\pi f)} \tag{11.10}$$

A function such as (11.9), which is the Fourier transform of a lag window, is called a *spectral window*. A function such as (11.10), which is the Fourier transform of a data window, is called a *frequency window*. The forms of (11.9) and (11.10) differ from the form of (11.4) due to the aliasing that occurs when the continuous-time window function is sampled to obtain a discrete-time window.

11.3 Triangular Window

A simple, but not particularly high-performance, window is the *triangular window* shown in Fig. 11.17 and is defined by

$$w(t) = 1 - \frac{2|t|}{\tau} \qquad |t| \le \frac{\tau}{2} \tag{11.11}$$

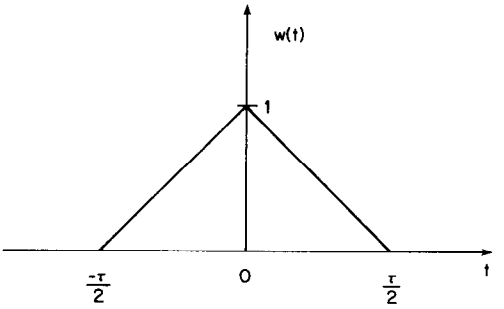

Figure 11.17 Triangular window.

Window functions are almost always even symmetric, and it is customary to show only the positive-time portion of the window as in Fig. 11.18. The triangular window is sometimes called the *Bartlett window* after M. S. Bartlett who described its use in a 1950 paper (Bartlett 1950). The Fourier transform of Eq. (11.11) is given by

$$W(f) = \frac{\tau}{2}\left[\frac{\sin(\pi f\tau/2)}{(\pi f\tau/2)}\right]^2 \tag{11.12}$$

The magnitude of (11.12) is plotted in Fig. 11.19. The peaks of the first through fourth sidelobes are attenuated by 26.5, 35.7, 41.6, and 46.0 dB,

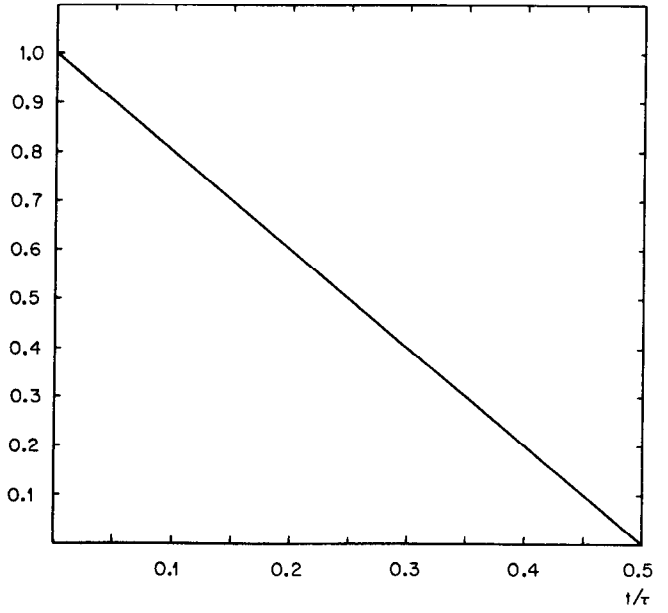

Figure 11.18 One-sided plot of a triangular window.

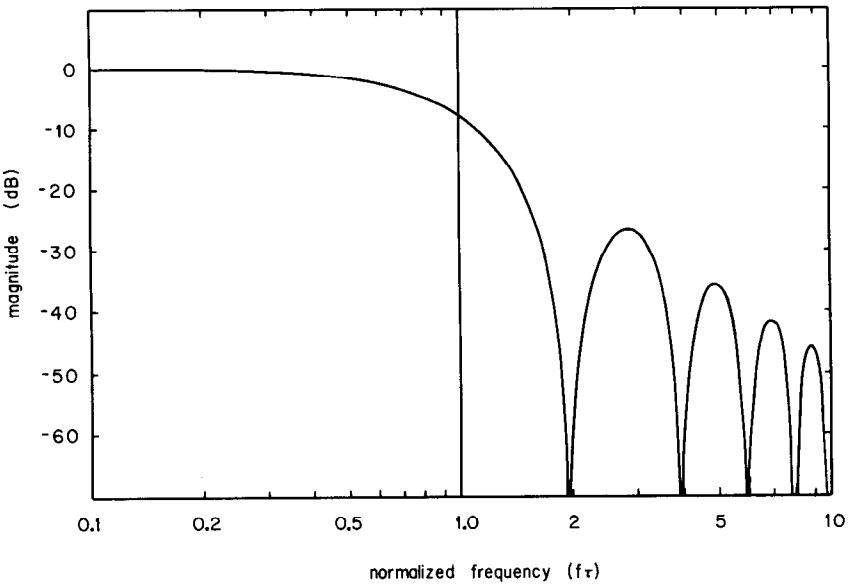

Figure 11.19 Magnitude response of a continuous-time triangular window.

respectively. The data for Fig. 11.19 was generated using the C function **contTriangularResponse( )** provided in Listing 11.7.

Discrete-time triangular window

If the function defined by (11.11) is sampled using $N = 2M + 1$ samples with $\tau = 2MT$, one sample at $t = 0$, and samples at nT for $n = \pm 1, \pm 2, \ldots, \pm M$, the sampled window function becomes the lag window defined by

$$w[n] = 1 - \frac{2|n|}{2M} \qquad -M \le n \le M \tag{11.13}$$

for the normalized case of $T = 1$. This equation can be expressed in terms of the total number of samples N by substituting $(N-1)/2$ for M to obtain

$$w[n] = 1 - \frac{2|n|}{N-1} \qquad \frac{-(N-1)}{2} \le n \le \frac{N-1}{2} \tag{11.14}$$

In some texts (such as Marple 1987 and Kay 1988), Eq. (11.14) is given as the definition of the discrete-time triangular window. However, evaluation of this equation reveals that $w[n] = 0$ for $n = \pm[(N-1)/2]$. This means that the two endpoints do not contribute to the window contents and that the window length is effectively reduced to $N - 2$ samples. In order to maintain a total of

N *nonzero* samples, many authors substitute $(N+2)$ for N in Eq. (11.14) to obtain

$$w[n] = 1 - \frac{|2n|}{N+1} \qquad \frac{-(N-1)}{2} \le n \le \frac{N-1}{2} \qquad (11.15)$$

N odd

For an even number of samples, the window values can be obtained by substituting $(n + \frac{1}{2})$ for n in Eq. (11.15) to obtain a window that is symmetrical about a line midway between $n = -1$ and $n = 0$. (The equals sign in the box below is in quotes because n can assume only integer values; nevertheless, n "=" $-\frac{1}{2}$ is a convenient shorthand way of saying "midway between $n = -1$ and $n = 0$.")

$$w[n] = 1 - \frac{|2n+1|}{N+1} \qquad \frac{-N}{2} \le n \le \frac{N}{2} - 1 \qquad (11.16)$$

N even, center at n "=" $\frac{-1}{2}$

Alternatively, we could substitute $(n - \frac{1}{2})$ for n in Eq. (11.15) to obtain a window symmetric about a line midway between $n = 0$ and $n = 1$:

$$w[n] = 1 - \frac{|2n-1|}{N+1} \qquad \frac{-N}{2} + 1 \le n \le \frac{N}{2} \qquad (11.17)$$

N even, center at n "=" $\frac{1}{2}$

An expression for the triangular *data* window can be obtained by substituting $[n - (N-1)/2]$ for n in Eq. (11.15) or by substituting $(n - N/2)$ for n in Eq. (11.16) to yield

$$w[n] = 1 - \frac{|2n-N+1|}{N+1} \qquad 0 \le n \le N-1 \qquad (11.18)$$

Section 11.4 will present several C functions for generating various forms of the discrete-time triangular window.

Frequency and spectral windows

The spectral window obtained from the DTFT of the lag window (11.14) is given by

$$W(f) = \frac{1}{M}\left[\frac{\sin(M\pi f)}{\sin(\pi f)}\right]^2 \qquad (11.19a)$$

or
$$W(\theta) = \frac{2}{N}\left\{\frac{\sin[(N/4)\theta]}{\sin[(1/2)\theta]}\right\}^2 \tag{11.19b}$$

where $M = \dfrac{N-1}{2}$

$$\theta = \frac{2\pi f}{f_s}$$

The form of (11.19) is closely related to the Fejer kernel $F_n(\cdot)$, which, like the Dirichlet kernel presented in Sec. 11.3, has some variety in its definition:

$$F_n(x) \triangleq \frac{\sin^2(n\pi x)}{n \sin^2(\pi x)}$$

(Priestley 1981)

$$F_n(\theta) \triangleq \frac{\sin^2(n\theta/2)}{2\pi n \sin^2(\theta/2)}$$

(Dym and McKean 1972)

The magnitude of (11.19) for $N = 11$ and $N = 21$ is plotted in Fig. 11.20. The data for these plots were obtained using the C function **discTriangularResponse( )** provided in Listing 11.8.

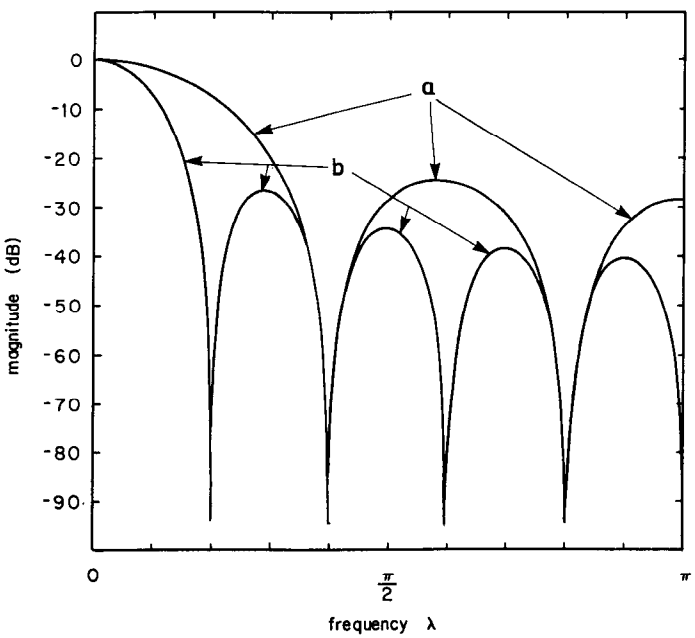

Figure 11.20 Magnitude of the DTFT for (*a*) an 11-point triangular window and (*b*) a 21-point triangular window.

11.4 Window Software

As we saw in the previous section, a window function can come in a number of different varieties—odd-length lag window, even-length lag window centered on $n = \frac{1}{2}$, and so on. As was done for the triangular window, an explicit function for each variety can be derived. However, the task of designing and coding computer programs to generate window coefficients can be simplified somewhat if we view the different varieties from a slightly different perspective. Despite the apparent variety of specific formats, there are really only two basic forms that need to be generated—one form for odd-length windows and one form for even-length windows. All of the specific varieties can be generated as simply horizontal translations of these two forms. Furthermore, since all the windows considered in this book are symmetric, we need to generate the coefficients for only half of each window. An odd-length lag window is probably the most "natural" of the discrete-time windows. Consider the triangular window shown in Fig. 11.21, which has sample values indicated at $t = \pm nT$ for $n = 0, 1, 2, \ldots$. Because of symmetry, we will require our program to generate the $(N+1)/2$ coefficients corresponding to $t = 0, T, 2T, 3T, \ldots, (N-1)T/2$ and place them in locations 0 through $(N-1)/2$ of an array called **window[ ]**. These coefficients can be obtained using Eq. (11.15). Next we consider the triangular window shown in Fig. 11.22. This window has been shifted so that its axis of symmetry lies at $t = -T/2$. The sample values indicated in the figure can be obtained from Eq. (11.16). The sample values for either the even-length case of Fig. 11.22 or the odd-length case of Fig. 11.21 can be obtained from the combined formula

$$w[n] = 1 - \frac{2|x|}{N+1}$$

where $x = \begin{cases} n & \text{for } N \text{ odd} \\ n + \frac{1}{2} & \text{for } N \text{ even} \end{cases}$

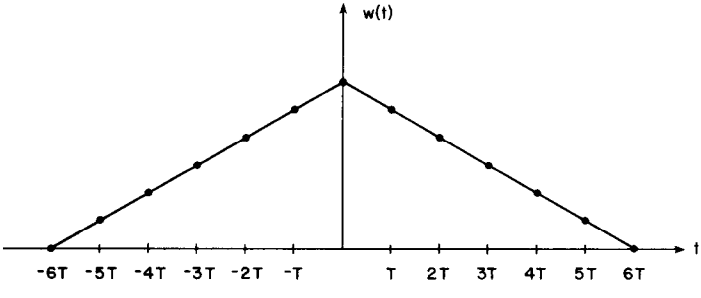

Figure 11.21 Triangular window sampled to produce an odd-length lag window.

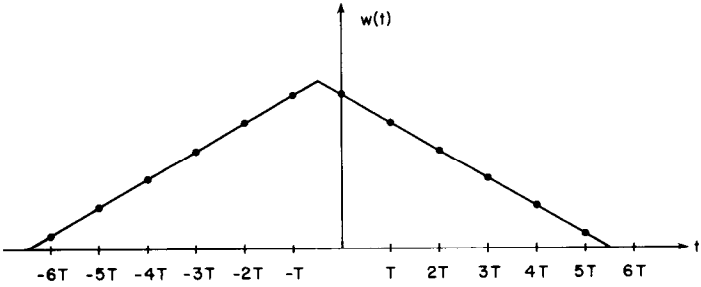

Figure 11.22 Triangular window shifted and sampled to produce an even-length lag window with axis of symmetry midway between $n = -1$ and $n = 0$.

The C function, **triangularWindow( )**, provided in Listing 11.9, uses this formula to generate coefficients for both odd- and even-length triangular windows. For N odd, the value returned in **window[0]** lies on the full window's axis of symmetry and is the value of the continuous-time window at $t = 0$. For N even, the value returned in **window[0]** lies one-half sample-time to the right of the full window's axis of symmetry and is the value of the continuous-time window at $t = T/2$.

Generating and storing a complete lag window would be conceptually straightforward if C allowed the use of negative indices for arrays. Although it is not possible to define an array that takes negative indices, it is possible to give the appearance of negative indices by using the special structure called **WWWW**, which is defined by the following code fragment:

```
typedef struct{
                real left[256];
                real right[256];
                } timeRecord;
union timeRec{
                real full[512];
                timeRecord half;
                } WWWW;
   # define LAG_WINDOW WWWW.half.right
   # define DATA_WINDOW WWWW.full
```

Use of this special structure is one way to permit negative index values for an array. The array **WWWW.half.right** can take a negative index because of the space reserved by the **left[ ]** array within the structure half of type **timeRecord**. The macro **LAG_WINDOW** is defined to facilitate easier reference to **WWWW.half.right**. For example, the C statement

```
LAG_WINDOW[5] = 0.735;
```

will place the value 0.735 into location 5 of the array **WWWW.half.right** (which, owing to the union, is also location 261 of the array **WWWW.full** or

DATA_WINDOW). The statements

```
LAG_WINDOW[-1] = 0.25;
LAG_WINDOW[-256] = 0.5;
```

will place the value 0.25 into location 255 and the value 0.5 into location 0 of the array **WWWW.half.left**.

The C function **makeLagWindow( )**, provided in Listing 11.10, takes a half window as generated by **triangularWindow( )** (or similar functions for other window shapes to be presented in subsequent sections) and converts it into a full lag window. For the output array, the call of this function should use the array **WWWW.half.right** or its defined alias **LAG_WINDOW**:

```
makeLagWindow( numbTaps, window, center, LAG_WINDOW);
```

If N is odd, the full window will be placed in locations $-(N-1)/2$ through $(N-1)/2$ of the "array" **LAG_WINDOW[ ]**. If N is even and **center** is negative, the full window will be placed in locations $-N/2$ through $(N/2)-1$ of **LAG_WINDOW[ ]**. If N is even and **center** is positive, the full window will be placed in locations $-(N/2)+1$ through $N/2$ of **LAG_WINDOW[ ]**.

The C function **makeDataWindow( )**, provided in Listing 11.11, takes a half window as generated by **triangularWindow( )** (or similar functions) and converts it into a full data window. For the output array, the call of this function should use the array **WWWW.full** or its defined alias **DATA_WINDOW**—

```
makeDataWindow( numbTaps, window, DATA_WINDOW);
```

If N is odd, the input value **window[0]** will lie on the axis of symmetry of the output in **DATA_WINDOW[ ]**. If N is even, the input value **window[0]** will appear in two consecutive locations in the center of the output window, and the axis of symmetry will lie between these two locations.

11.5 Applying Windows to Fourier Series Filters

Conceptually, a tapering window such as the triangular window is applied to the input of an FIR approximation to an ideal filter. However, since multiplication is associative, a much more computationally efficient implementation can be had by multiplying the window coefficients and the original filter coefficients to arrive at a modified set of filter coefficients. The impulse response coefficients produced by the C functions of Sec. 11.1 are generated in a data window format {that is, $h[n]$ is defined for $0 \le n \le N-1$}. Therefore the window coefficients should also be put into a data window format before multiplying them with the ideal filter coefficients of Sec. 11.1.

TABLE 11.5 Coefficients for a 21-tap Lowpass Filter
{$h[n]$ are the original coefficients; $w[n]$ are triangular window coefficients}

n	$h[n]$	$w[n]$	$w[n] \cdot h[n]$
0, 20	0.000000	0.000000	0.000000
1, 19	−0.033637	0.090909	−0.006116
2, 18	−0.023387	0.181818	−0.006378
3, 17	0.026728	0.272727	0.009719
4, 16	0.050455	0.363636	0.022934
5, 15	0.000000	0.454545	0.000000
6, 14	−0.075683	0.545455	−0.048162
7, 13	−0.062366	0.636364	−0.045357
8, 12	0.093549	0.727273	0.076540
9, 11	0.302731	0.909091	0.275210
10	0.400000	1.00	0.400000

Example 11.5 Apply a triangular window to the 21-tap lowpass filter of Example 11.1.

solution Table 11.5 lists the original values of the filter coefficients, the corresponding discrete-time window coefficients, and the final values of the filter coefficients after the windowing has been applied. The frequency response of the windowed filter is shown in Figs. 11.23 and 11.24. The response looks pretty good when plotted against a linear axis

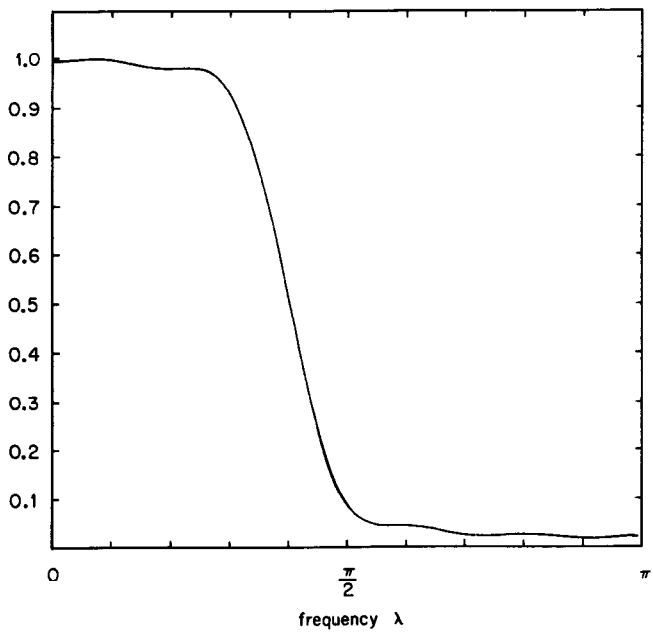

Figure 11.23 Magnitude response (as a percentage of peak) for a triangular-windowed 21-tap lowpass filter.

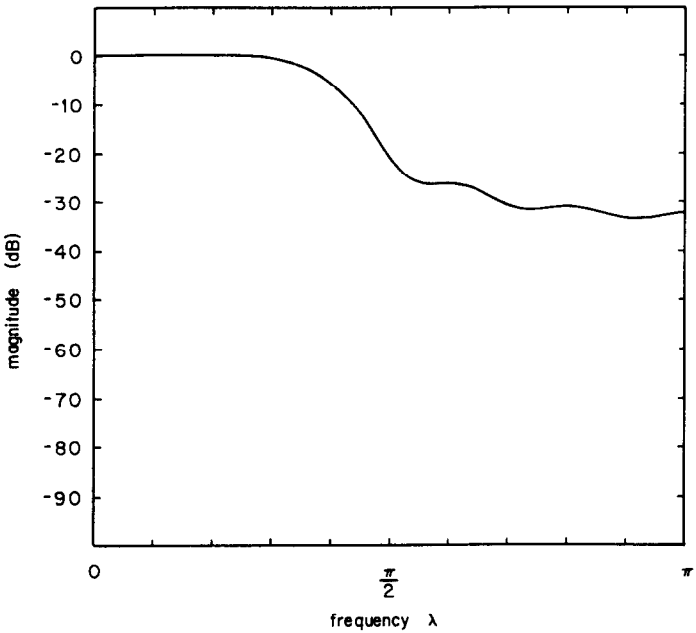

Figure 11.24 Magnitude response (in decibels) for a triangular-windowed 21-tap lowpass filter.

as in Fig. 11.23, but the poor stop-band performance is readily apparent when the response is plotted on a decibel scale as in Fig. 11.24.

11.6 von Hann Window

The continuous-time von Hann window function shown in Fig. 11.25 is defined by

$$w(t) = 0.5 + 0.5 \cos \frac{2\pi t}{\tau} \qquad |t| \le \frac{\tau}{2} \tag{11.20}$$

The corresponding frequency response, shown in Fig. 11.26, is given by

$$W(f) = 0.54\tau \operatorname{sinc}(\pi f \tau) + 0.23\tau \operatorname{sinc}[\pi\tau(f-\tau)] + 0.23\tau \operatorname{sinc}[\pi\tau(f+\tau)] \tag{11.21}$$

The first sidelobe of this response is 31.4 dB below the main lobe, and the main lobe is twice as wide as the main lobe of the rectangular window. References to the von Hann window as the "hanning" window are widespread throughout the signal processing literature. This is unfortunate for two reasons. First, the window gets its name from Julius von Hann, *not* some nondescript Mr. Hanning. Second, the term *hanning* is easily (and often) confused with *Hamming*. Oppenheim and Schafer (1975) insinuate that the

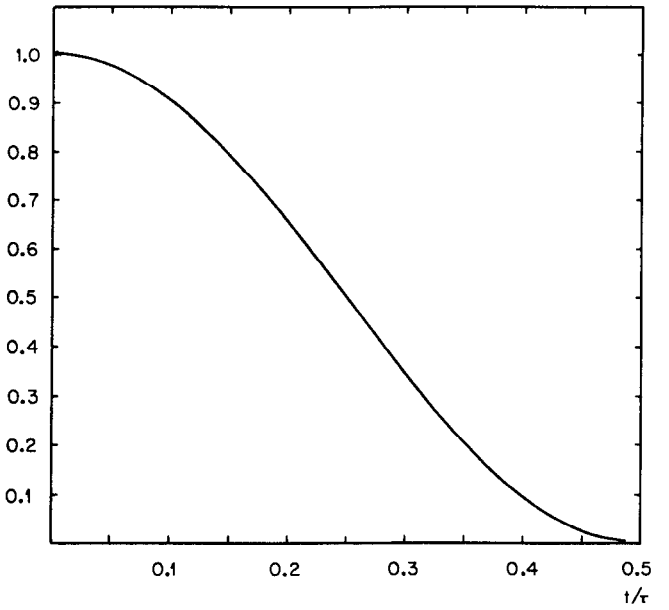

Figure 11.25 The von Hann window.

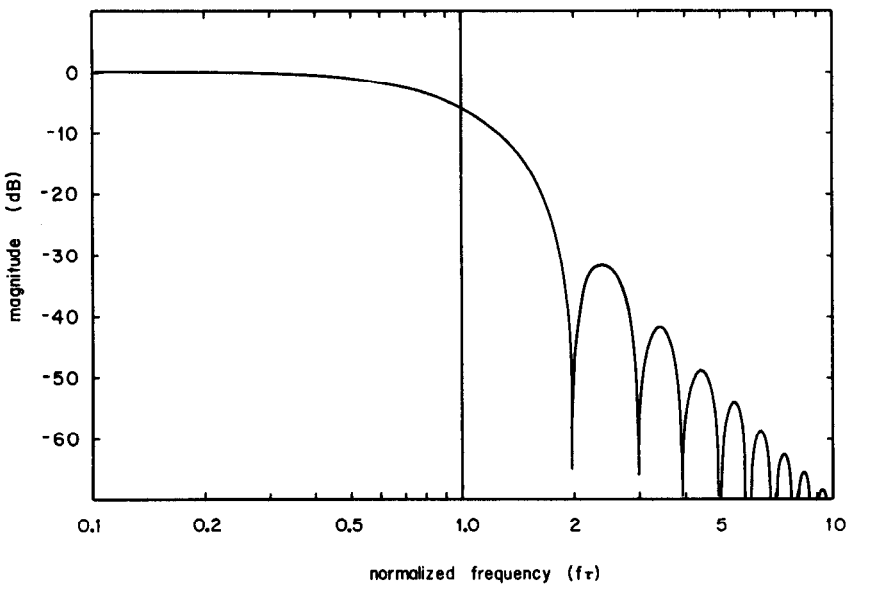

Figure 11.26 Magnitude response of the von Hann window.

incorrect use of *hanning* is due to Blackman and Tukey (1958). This window is ocassionally called a *raised-cosine window.*

Discrete-time von Hann window

If the function defined by Eq. (11.20) is sampled using $N = 2M + 1$ samples with one sample at $t = 0$ and samples at nT for $n = \pm 1, \pm 2, \ldots, \pm M$, the sampled window function becomes

$$w[n] = 0.5 + 0.5 \cos \frac{\pi n}{M} \qquad -M \leq n \leq M \tag{11.22}$$

for the normalized case of $T = 1$. Evaluation of (11.22) reveals that $w(n) = 0$ for $n = \pm M$. This means that the two endpoints do not contribute to the window contents and that the window length is effectively reduced to $N - 2$ samples. In order to mantain a total of N *nonzero* samples, we must substitute $M + 1$ for M in Eq. (11.22) to yield

$$w[n] = 0.5 + 0.5 \cos \frac{2\pi n}{2(M+1)} \qquad -M \leq n \leq M \tag{11.23}$$

Equation (11.23) can now be recast in terms of N by substituting $(N-1)/2$ for M to obtain

$$w[n] = 0.5 + 0.5 \cos \frac{2\pi n}{N-1} \qquad \frac{-(N-1)}{2} \leq n \leq \frac{N-1}{2} \tag{11.24}$$

n odd

For an even number of samples, the window values can be obtained by substituting either $(n + \frac{1}{2})$ or $n(-\frac{1}{2})$ for n in Eq. (11.24) to obtain

$$w[n] = 0.5 + 0.5 \cos \frac{\pi(2n+1)}{N-1} \qquad \frac{-N}{2} \leq n \leq \frac{N}{2} - 1 \tag{11.25}$$

N even, center at n "=" $\frac{-1}{2}$

$$w[n] = 0.5 + 0.5 \cos \frac{\pi(2n-1)}{N-1} \qquad \frac{-N}{2} + 1 \leq n \leq \frac{N}{2} \tag{11.26}$$

N even, center at n "=" $\frac{1}{2}$

The C function, **hannWindow( )**, provided in Listing 11.12, generates coefficients for the von Hann window.

TABLE 11.6 Coefficients for a 21-tap Lowpass Filter
{$h[n]$ are the original coefficients; $w[n]$ are von Hann window coefficients}

n	$h[n]$	$w[n]$	$w[n] \cdot h[n]$
0, 20	0.000000	0.000000	0.000000
1, 19	−0.033637	0.024472	−0.000823
2, 18	−0.023387	0.095492	−0.002233
3, 17	0.026728	0.206107	0.005509
4, 16	0.050455	0.345492	0.017432
5, 15	0.000000	0.500000	0.000000
6, 14	−0.075683	0.654508	−0.049535
7, 13	−0.062366	0.793893	−0.049512
8, 12	0.093549	0.904508	0.084616
9, 11	0.302731	0.975528	0.295323
10	0.400000	1.00	0.400000

Example 11.6 Apply a von Hann window to the 21-tap lowpass filter of Example 11.1.

solution Table 11.6 lists the original values of the filter coefficients, the corresponding discrete-time window coefficients, and the final values of the filter coefficients after the windowing has been applied. The frequency response of the windowed filter is shown in Fig. 11.27.

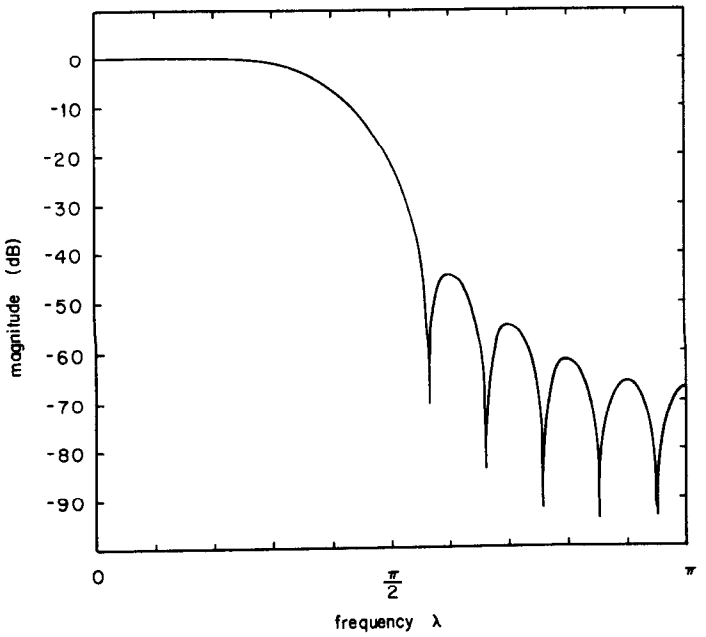

Figure 11.27 Magnitude response for a von Hann-windowed 21-tap lowpass filter.

11.7 Hamming Window

The continuous-time Hamming window function shown in Fig. 11.28 is defined by

$$w(t) = 0.54 + 0.46 \cos\frac{2\pi t}{\tau} \qquad |t| \le \frac{\tau}{2} \tag{11.27}$$

The Fourier transform of Eq. (11.27) is given by

$$W(f) = 0.54\tau \operatorname{sinc}(\pi f\tau) + 0.23\tau \operatorname{sinc}[\pi\tau(f-\tau)] + 0.23\tau \operatorname{sinc}[\pi\tau(f+\tau)] \tag{11.28}$$

The magnitude of (11.28) is plotted in Fig. 11.29. The highest sidelobe of this response is 42.6 dB below the main lobe, and the main lobe is twice as wide as the main lobe of the rectangular window's response. This window gets its name from R. W. Hamming, a pioneer in the areas of numerical analysis and signal processing, who opened his numerical analysis text (Hamming 1972) with the now famous and oft-quoted pearl, "The purpose of computing is insight, not numbers."

Discrete-time Hamming windows

If the function defined by Eq. (11.27) is sampled using $N = 2M + 1$ samples with one sample at $t = 0$ and samples at nT for $n = \pm 1, \pm 2, \ldots, \pm M$, the

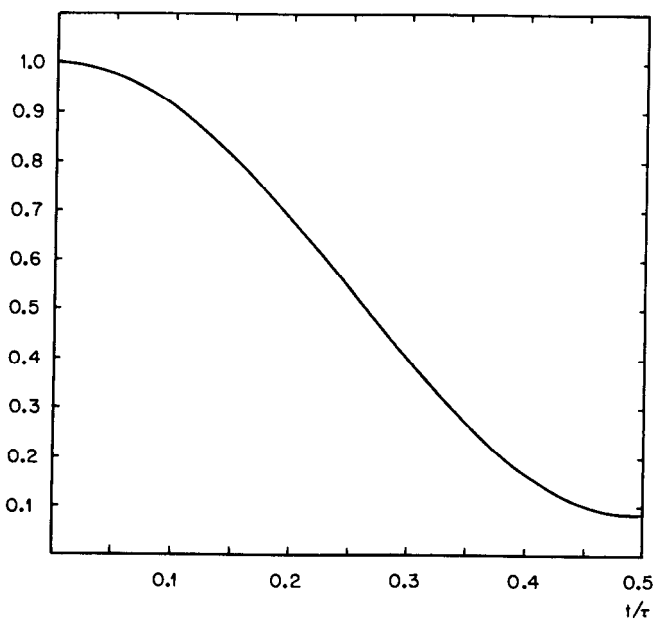

Figure 11.28 Hamming window.

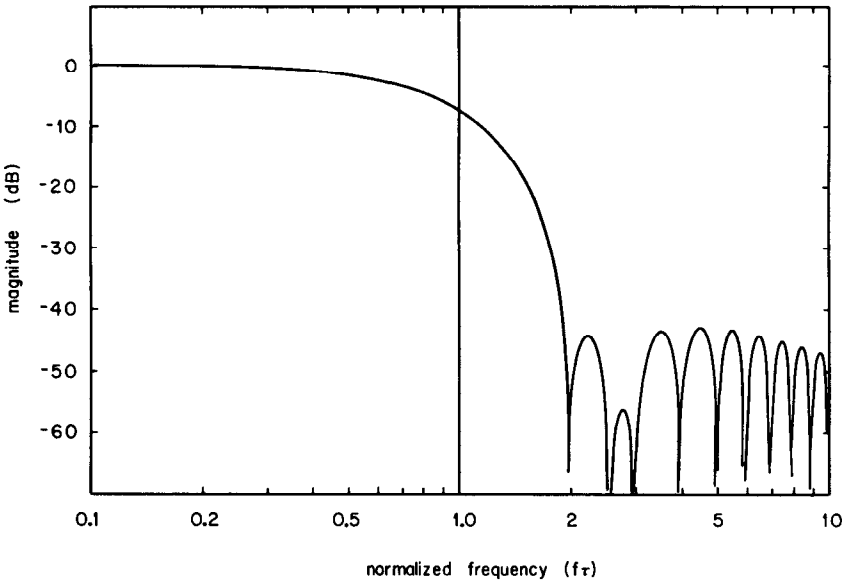

Figure 11.29 Magnitude response of the Hamming window.

sampled window function becomes the lag window defined by

$$w[n] = 0.54 + 0.46 \cos \frac{2\pi n}{2M} \qquad -M \le n \le M \tag{11.29}$$

for the normalized case of $T = 1$. Equation (11.29) can be expressed in terms of the total number of samples N by substituting $(N-1)/2$ for M to obtain

$$w[n] = 0.54 + 0.46 \cos \frac{2\pi n}{N-1} \qquad \frac{-(N-1)}{2} \le n \le \frac{N-1}{2} \tag{11.30}$$

n odd

For an even number of samples, the window values can be obtained by substituting $n + \frac{1}{2}$ for n in Eq. (11.30) to obtain

$$w[n] = 0.54 + 0.46 \cos \frac{\pi(2n+1)}{N-1} \qquad \frac{-N}{2} \le n \le \frac{N}{2} - 1 \tag{11.31}$$

N even, center at n "$=$" $\dfrac{-1}{2}$

The data window form can be obtained by substituting $[n - (N-1)/2]$ for n in

Eq. (11.30) or by substituting $(n - N/2)$ for n in Eq. (11.31) to yield

$$w[n] = 0.54 - 0.46 \cos \frac{2\pi n}{N-1} \qquad 0 \le n \le N-1 \qquad (11.32)$$

(Note the change in sign for the cosine term—this is *not* a typographical error.)

Example 11.7 Apply a Hamming window to the 21-tap lowpass filter of Example 11.1.

solution The windowed values of $h[k]$ are listed in Table 11.7, and the corresponding frequency response is shown in Fig. 11.30.

Computer generation of window coefficients

The C function **hammingWindow( )**, provided in Listing 11.13, generates ordinates for the Hamming window. The output conventions for even and odd N are as described in Sec. 11.4.

11.8 Dolph-Chebyshev Window

The Dolph-Chebyshev window is somewhat different from the other windows in this chapter in that a closed-form expression for the time domain window is not known. Instead, this window is defined as the inverse Fourier transform of the sampled-frequency response which is given by

$$W[k] = (-1)^k \frac{\cos\{N \cos^{-1}[\beta \cos(\pi k/N)]\}}{\cosh(N \cosh^{-1} \beta)} \qquad -(N-1) \le k \le N-1 \qquad (11.33)$$

A sidelobe level of -80 dB is often claimed for this response, but in fact, Eq. (11.33) defines a family of windows in which the minimum stop-band attenuation is a factor of β. A stop-band attenuation of 20α dB is obtained for a value

TABLE 11.7 Coefficients for a 21-tap Hamming-Windowed Lowpass Filter

k	$h[k]$
0, 20	0.000000
1, 19	−0.003448
2, 18	−0.003926
3, 17	0.007206
4, 16	0.020074
5, 15	0.000000
6, 14	−0.051627
7, 13	−0.050540
8, 12	0.085330
9, 11	0.295915
10	0.400000

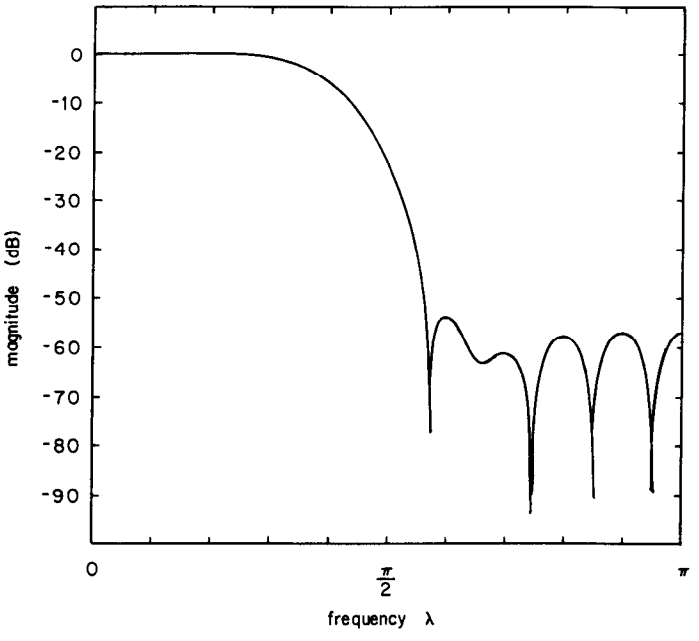

Figure 11.30 Magnitude response for a Hamming-windowed 21-tap lowpass filter.

of β given by

$$\beta = \cosh\left[\frac{1}{N}\cosh^{-1}(10^{\alpha})\right] \tag{11.34}$$

Often, $\beta > 1$ and consequently, evaluation of (11.32) may entail taking the inverse cosine of values with magnitudes greater than unity. In such cases, the following formula can be used:

$$\cos^{-1} x = \begin{cases} \dfrac{\pi}{2} - \tan^{-1}\left(\dfrac{x}{\sqrt{1-x^2}}\right) & |x| < 1 \\ \ln(x + \sqrt{x^2 - 1}) & |x| \geq 1 \end{cases} \tag{11.35}$$

The Dolph-Chebyshev window takes its name from C. L. Dolph and Pafnuti Chebyshev. The function $W(k)$ is a normalized form of the function developed by Dolph (1946) for specifying an antenna pattern optimized to achieve a narrow main lobe while simultaneously restricting the sidelobe response. Helms (1968) applied Dolph's result to the analogous problem of optimizing a filter response for a narrow transition band while simultaneously restricting sidelobe response. The name of Chebyshev is associated with this window because for integer values of z, the numerator of Eq. (11.33) is the zth order Chebyshev polynomial:

$$T_z(x) \equiv \cos(z \cos^{-1} x)$$

Listing 11.1 idealLowpass( )

```
/***********************************/
/*                                 */
/*   Listing 11.1                  */
/*                                 */
/*   idealLowpass()                */
/*                                 */
/***********************************/

void idealLowpass(    int numbTaps,
                      real lambdaU,
                      real hh[])
{
int n,nMax;
real mm;
printf("in idealLowpass\n");

for( n=0; n<numbTaps; n++)
    {
    mm = n - (real)(numbTaps-1)/2.0;
    if(mm==0)
        {hh[n] = lambdaU/PI;}
    else
        {hh[n] = sin(mm * lambdaU)/(mm * PI);}
    }
return;
}
```

Listing 11.2 idealHighpass( )

```
/***********************************/
/*                                 */
/*   Listing 11.2                  */
/*                                 */
/*   idealHighpass()               */
/*                                 */
/***********************************/

void idealHighpass( int numbTaps,
                      real lambdaL,
                      real hh[])
{
int n,nMax;
real mm;
printf("in idealHighpass\n");
```

```
for( n=0; n<numbTaps; n++)
    {
    mm = n - (real)(numbTaps-1)/2.0;
    if(mm==0)
        {hh[n] = 1.0 - lambdaL/PI;}
    else
        {hh[n] = -sin(mm * lambdaL)/(mm * PI);}
    }
return;
}
```

Listing 11.3 idealBandpass( )

```
/**********************************/
/*                                */
/*   Listing 11.3                 */
/*                                */
/*   idealBandpass()              */
/*                                */
/**********************************/

void idealBandpass( int numbTaps,
                    real lambdaL,
                    real lambdaU,
                    real hh[])
{
int n,nMax;
real mm;
printf("in idealBandpass\n");

for( n=0; n<numbTaps; n++)
    {
    mm = n - (real)(numbTaps-1)/2.0;
    if(mm==0)
        {hh[n] = (lambdaU - lambdaL)/PI;}
    else
        {hh[n] = (sin(mm * lambdaU) - sin(mm * lambdaL))/(mm * PI);}
    }
return;
}
```

Listing 11.4 idealBandstop( )

```
/***********************************/
/*                                 */
/*   Listing 11.4                  */
/*                                 */
/*   idealBandstop()               */
/*                                 */
/***********************************/

void idealBandstop( int numbTaps,
                    real lambdaL,
                    real lambdaU,
                    real hh[])
{
int n,nMax;
real mm;
printf("in idealBandstop\n");

for( n=0; n<numbTaps; n++)
    {
    mm = n - (real)(numbTaps-1)/2.0;
    if(mm==0)
        {hh[n] = 1.0 + (lambdaL - lambdaU)/PI;}
    else
        {hh[n] = (sin(n * lambdaL) - sin(mm * lambdaU))/(mm * PI);}
    }
return;
}
```

Listing 11.5 contRectangularResponse( )

```
/***********************************/
/*                                 */
/*   Listing 11.5                  */
/*                                 */
/*   contRectangularResponse()     */
/*                                 */
/***********************************/

#define TINY 3.16e-5

real contRectangularResponse( real freq, real tau, logical dbScale)
{
real x;
```

```
x = sinc(PI * freq * tau);
if(dbScale)
    {
    if(fabs(x) < TINY)
        {x = -90.0;}
    else
        {x = 20.0*log10(fabs(x));}
    }
return(x);
}
```

Listing 11.6 discRectangularResponse( )

```
/***********************************/
/*                                 */
/*   Listing 11.6                  */
/*                                 */
/*   discRectangularResponse()     */
/*                                 */
/***********************************/


real discRectangularResponse( real freq,
                              int M,
                              logical normalizedAmplitude)
{
real result;

if(freq == 0.0)
    { result = (real) (2*M+1);}
else
    { result = fabs(sin(PI * freq * (2*M+1))/ sin( PI * freq));}

if( normalizedAmplitude ) result = result / (real) (2*M+1);
return(result);
}
```

Listing 11.7 contTriangularResponse( )

```
/***********************************/
/*                                 */
/*   Listing 11.7                  */
/*                                 */
/*   contTriangularResponse()      */
/*                                 */
/***********************************/


real contTriangularResponse(  real freq,
                              real tau,
                              logical dbScale)
{
real amp0, x;
amp0 = 0.5 * tau;
x = PI * freq * tau / 2.0;
x = 0.5 * tau * sincSqrd(x);
if(dbScale)
    {
    if(fabs(x/amp0) < TINY)
        {x = -90.0;}
    else
        {x = 20.0*log10(fabs(x/amp0));}
    }
return(x);
}
```

Listing 11.8 discTriangularResponse( )

```
/***********************************/
/*                                 */
/*   Listing 11.8                  */
/*                                 */
/*   discTriangularResponse()      */
/*                                 */
/***********************************/


real discTriangularResponse(  real freq,
                              int M,
                              logical normalizedAmplitude)
{
real result;
```

```
if(freq == 0.0)
    { result = (real) M;}
else
    { result = (sin(PI * freq * M) * sin(PI * freq * M)) /
                (M * sin( PI * freq) * sin( PI * freq));
    }

if( normalizedAmplitude ) result = result / (real) M;
return(result);
}
```

Listing 11.9 triangularWindow( )

```
/***********************************/
/*                                 */
/*   Listing 11.9                  */
/*                                 */
/*   triangularWindow()            */
/*                                 */
/***********************************/

void triangularWindow( int N, real window[])
{
real offset;
int n;
offset = (real) (1-(N%2));

for(n=0; n<(N/2.0); n++)
    {
    window[n] = 1.0 - (2.0*n + offset)/(N+1.0);
    }
return;
}
```

Listing 11.10 makeLagWindow( )

```
/***********************************/
/*                                 */
/*   Listing 11.10                 */
/*                                 */
/*   makeLagWindow()               */
/*                                 */
/***********************************/

void makeLagWindow(   int N,
                      real window[],
                      int center,
                      real outWindow[])
```

```
{
int n,M;

if(N%2) {
    M=(N-1)/2;
    for(n=0; n<=M; n++) {
        outWindow[n] = window[n];
        outWindow[-n] = outWindow[n];
        }
    }
else {
    M=(N-2)/2;
    if(center == negative) {
        for( n=0; n<=M; n++) {
            outWindow[n] = window[n];
            outWindow[-(1+n)] = window[n];
            }
        }
    else {
        for( n=0; n<=M; n++) {
            outWindow[n+1] = window[n];
            outWindow[-n] = window[n];
            }
        }
    }
return;
}
```

Listing 11.11 makeDataWindow( )

```
/**********************************/
/*                                */
/*   Listing 11.11                */
/*                                */
/*   makeDataWindow()             */
/*                                */
/**********************************/


void makeDataWindow( int N,
                     real window[],
                     real outWindow[])
{
int n,M;
```

```
if(N%2) {
    M=(N-1)/2;
    for(n=0; n<=M; n++) {
        outWindow[n] = window[M-n];
        outWindow[M+n] = window[n];
        }
    }
else {
    M=(N-2)/2;
    for(n=0; n<=M; n++) {
        outWindow[n] = window[M-n];
        outWindow[M+n+1] = window[n];
        }
    }
return;
}
```

Listing 11.12 hannWindow( )

```
/***********************************/
/*                                 */
/*   Listing 11.12                 */
/*                                 */
/*   hannWindow()                  */
/*                                 */
/***********************************/
void hannWindow( int N, real window[])
{
logical odd;
int n;
odd = N%2;

for(n=0; n<(N/2.0); n++)
    {
    if( odd)
        {window[n] = 0.5 + 0.5 * cos(TWO_PI*n/(N-1));}
    else
        {window[n] = 0.5 + 0.5 * cos(TWO_PI * (2*n+1)/(2.0*(N-1)));}
    }
return;
}
```

Listing 11.13 hammingWindow( )

```
/**********************************/
/*                                */
/*   Listing 11.13                */
/*                                */
/*   hammingWindow()              */
/*                                */
/**********************************/


void hammingWindow( int N, real window[])
{
logical odd;
int n;
odd = N%2;

for(n=0; n<(N/2.0); n++)
    {
    if( odd)
        {window[n] = 0.54 + 0.46 * cos(TWO_PI*n/(N-1));}
    else
        {window[n] = 0.54 + 0.46 *
                     cos(TWO_PI * (2*n+1)/(2.0*(N-1)));}
    }
return;
}
```