CHAPTER

8

Analysis of Discrete Linear Systems—The z-Transform and Applications to Filters

8.1 GOALS OF THIS CHAPTER

The transfer function, in either the frequency or complex frequency (Laplace) domain, provides a complete description of the behavior of any continuous linear, time-invariant (LTI) system. Given the transfer function, or equivalently the impulse response, we can evaluate the response of the system to any stimulus. All physiological systems are continuous, and to the extent that they can be considered linear and time invariant, the techniques of Chapters 5–7 can be used to analyze these systems.

We have the tools to describe all continuous linear systems, but not discrete systems. The only discrete systems you are likely to encounter live in computers,[1] and of these, the only systems likely to be important to you are "digital filters." But they are important. Every signal you measure contains some noise and most would benefit from filtering. So you need to understand the basics of discrete systems analysis to intelligently use, and possibly design digital filters.

The digital filters described here are all linear, time-invariant, discrete-time systems, abbreviated "LTID." LTID systems can be analyzed using the discrete Fourier transform, but restrictions apply. Recall that to apply the Fourier transform methods to (continuous) LTI systems, we need to assume steady-state signals and that the system has no initial conditions. If these conditions are not met, the more general Laplace transform must be used. Additional requirements apply when using the discrete Fourier transform to analyze a discrete system: all signals must be summable (that is, they cannot be growing with time), and the system must meet certain stability requirements. Biomedical signals and digital filters are likely to meet these criteria, so the good old Fourier transform, specifically the fast Fourier transform (FFT), will be our primary tool for analyzing digital filters. Nonetheless, there is a more general approach to discrete system analysis, known as the "z-transform," and you need at least

[1]Or in computer-like devices such as field programmable gate arrays known as FPGAs.

Circuits, Signals, and Systems for Bioengineers
https://doi.org/10.1016/B978-0-12-809395-5.00008-4

a passing facility with this approach as z-transform notation is used for describing all discrete systems, including digital filters.

The z-transform is nothing more than a discrete version of the Laplace transform. It can be developed as an extension of the discrete Fourier transform in a manner that parallels the development of the Laplace transform from the continuous Fourier transform (Equations 7.3 and 7.4). With the z-transform, we have a formidable array of system and signal processing techniques. Table 8.1 summarizes these analytical tools and shows how the z-transform fits in with these other techniques.

This chapter covers the development of the z-transform and its application to digital filters, the only discrete systems you are likely to encounter. Specific topics include:

- Development of the z-transform equation, which closely parallels the development of the Laplace transform.
- Definition of the digital (i.e., discrete) transfer function. This z-transform plays the same role as the Laplace transfer function for discrete systems. It is a function of the complex discrete frequency variable, z.
- Difference equations: discrete-time representation of discrete systems.
- Spectrum of a digital transfer function. How to use the FFT to plot the spectrum given a digital transfer function.
- General properties associated with all filters: bandwidth and roll-off or slope characteristics.

TABLE 8.1 Summary of System and Signal Analysis Techniques

Analysis	Continuous	Discrete	Constraints
Signals Time domain	Autocorrelation Cross-correlation	Autocorrelation Cross-correlation	Finite signals
Signals Frequency domain	Fourier series analysis Fourier transform	Discrete Fourier series Discrete Fourier transform	Periodic or aperiodic Dirichlet conditions
Systems Time domain	Impulse response, $h(t)$ Convolution	Impulse response, $h[n]$ Convolution	Linear, time invariant (LTI) …or for discrete systems Linear, time invariant discrete-time (LTID)
Systems, restricted Frequency domain	Fourier transform Transfer function, $TF(\omega)$	Discrete Fourier transform Transform function, $TF[\omega]$	LTI, steady state, no transient signals …or for discrete systems LTID, stable systems, summable signals (not growing with time)
Systems, general Complex frequency domain	Laplace transform Laplace transfer function, $TF(s)$	z-transform z-transfer function, $TF[z]$	LTI …or for discrete systems LTID

- Definition and properties of digital filters, including finite impulse response (FIR) filters and infinite impulse response (IIR) filters.
- Designing FIR filters with and without MATLAB. Finding the filter coefficients (or filter weights) from the desired frequency characteristics for FIR filters.
- Designing IIR filter with MATLAB. Finding the filter coefficients (or filter weights) from the desired frequency characteristics for IIR filters using a MATLAB Toolbox.

8.2 THE Z-TRANSFORM

The z-transform is a discrete domain analogy to the Laplace transform. In fact, it can be viewed as just a different version of the Laplace transform. The transform provides us with a discrete transfer function, which is, again, analogous to the Laplace transfer function. The discrete transfer function gives us the same analytical power for discrete systems as the Laplace transform provides for continuous systems. Specifically, the transfer function is a complete description of the behavior (the input/output characteristics) of the system.

To develop a digital version of the Laplace transform, we recap the approach that was used to develop the Laplace transform. Recall the Laplace transform is just an extension of the Fourier transform that uses complex frequency s (where $s = \sigma + j\omega$) instead of ordinary frequency $j\omega$. Replacing the regular frequency term (i.e., $j\omega$) in the Fourier transform with complex frequency (i.e., s) leads to the Laplace transform, repeated here:

$$X(\sigma,\omega) = \int_0^\infty x(t)e^{-\sigma t}e^{-j\omega t}dt = \int_0^\infty x(t)e^{-st}dt \tag{8.1}$$

The addition of the $e^{-\sigma t}$ term in the Laplace transform ensures convergence for a wider range of signals that includes step-like signals. These signals do not converge in the Fourier transform equation, but the additional $e^{-\sigma}$ term added to the Laplace equation provides convergence provided σ takes an appropriate value.

To develop a discrete version of the Laplace transform, we start by putting the Laplace equation into discrete notation. This conversion is achieved by replacing the integral with a summation and substituting a sample number, n, for the time variable t. (Actually $t = nT_s$, but we assume that T_s is normalized to 1.0 for this development.) These modifications give:

$$X(\sigma,\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-\sigma n}e^{-j\omega n} \tag{8.2}$$

Next, define a new variable:

$$r = e^{-\sigma} \tag{8.3}$$

Substituting r for $e^{-\sigma}$ into Equation 8.2 gives:

$$X(r,\omega) = \sum_{n=-\infty}^{\infty} x[n]r^n e^{-j\omega n} \tag{8.4}$$

Equation 8.4 is a valid form for the z-transform, but commonly, the equation is simplified by defining another new complex variable, z. This is essentially the idea used in the Laplace transform where the complex variable s is introduced to represent $\sigma + j\omega$. The new variable, z, is defined as:

$$z = r\,e^{j\omega} = |z|e^{j\omega} \tag{8.5}$$

where r is now simply the magnitude of the new complex variable z. Substituting z into Equation 8.4 gives:

$$X(z) = Z[x[n]] = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{8.6}$$

This is the defining equation for the z-transform of a digital signal $x[n]$ and is notated as $Z[x[n]]$. This is called the "bilateral z-transform" as the summation limits extend to $\pm\infty$. As defined in Section 1.4.3, causal signals are those that exist only for $t \geq 0$. For discrete causal signals, this translates to $n \geq 0$. Thus for discrete causal signals, the summation in Equation 8.6 only needs to be from $n = 0$ to ∞. This gives rise to the "unilateral z-transform" that is commonly used in digital systems analysis:

$$X(z) = Z[x[n]] = \sum_{n=0}^{\infty} x[n]z^{-n} \tag{8.7}$$

In any real application, the limit of the summation is finite, usually the length of the signal $x[n]$.

As with the Laplace transform, the z-transform is based on a complex variable: in this case the arbitrary complex variable z, which equals $|z|e^{j\omega}$. Analogous to Laplace variable s, the complex frequency, z, is the "discrete complex frequency." As with the Laplace variable s, it is possible to substitute $e^{j\omega}$ for z to perform a strictly sinusoidal analysis.[2] For us, this is a very useful property, as it allows us to easily determine the frequency characteristic of a z-transform transfer function using only the Fourier transform.

8.2.1 The Unit Delay

Multiplying a data sequence by z^{-n} in the z-domain has the useful effect of shifting the sequence by n data samples in the time domain. For example, assume you have a data sequence $x[n]$ that has a z-transform $X[z]$. Define another data sequence, $y[n]$, that is the same sequence but shifted over one position: $y[n] = x[n-1]$. The z-transform of this shifted sequence is by reference to Equation 8.7 is:

$$Y(z) = \sum_{n=0}^{\infty} y[n]z^{-n} = \sum_{n=0}^{\infty} x[n-1]z^{-n}$$

[2]If $|z|$ is set to 1, then from Equation 8.5, $z = e^{j\omega}$. This is called evaluating z on the "unit circle" because $|z| = 1$ describes a circle of radius one when z is plotted in polar coordinates. Substituting $e^{j\omega}$ for z is a useful operation because in digital filter analysis, our primary interest is the filter's spectrum.

If $k = n - 1$, then $x[n-1] = x[k]$ and $Y(z)$ becomes:

$$Y(z) = \sum_{k=0}^{\infty} x[k]z^{-(k+1)} = \sum_{k=0}^{\infty} x[k]z^{-k}z^{-1} = z^{-1}\sum_{k=0}^{\infty} x[k]z^{-k}$$

But the summation:

$$\sum_{k=0}^{\infty} x[k]z^{-k} = X(z)$$

So $Y(z)$ due to a shift in x of one sample becomes:

$$Y(z) = z^{-1}X(z) \tag{8.8}$$

This time-shifting property generalizes to higher powers of z, so multiplication by z^k in the z-domain is equivalent to a shift of k time samples:

$$Z[x[n-k]] = z^{-k}Z[x[n]] \tag{8.9}$$

For a discrete variable, $x[n] = [x_1, x_2, x_3, \ldots x_N]$, the power of z in the z-domain can be used to define a sample's position in the sequence. This is useful in understanding the z-domain transfer function.

8.2.2 The Digital Transfer Function

As in Laplace transform analysis, the z-transform allows us to define a digital transfer function. By analogy to previous transfer functions, the digital transfer function is defined as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{Z[y[n]]}{Z[x[n]]} \tag{8.10}$$

where $X(z)$ is the z-transform of the input signal, $x[n]$, and $Y(z)$ is the z-transform of the system's output, $y[n]$. As an example, a simple linear element known as a "unit delay" is shown in Figure 8.1. In this system, the time shifting characteristic of z^{-n} (Equations 8.8 and 8.9) is used to define processes where the output is the same as the input but shifted (or delayed) by one data sample. The z-transfer function for this process in Figure 8.1 is $H(z) = z^{-1}$.

The unit delay also generalizes to a higher power of n so if $H(z) = z^{-3}$ then $y[n] = x[n-3]$. The z-transform of the delayed version of $x[n]$ is z raised to the power of the delay and vice versa:

$$x[n-k] \Leftrightarrow z^{-k} \tag{8.11}$$

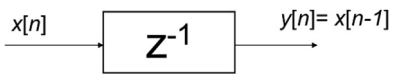

FIGURE 8.1 A unit delay is a linear system that shifts the input by one data sample. Other powers of z can be used to provide larger shifts. The transfer function for this system is $Y(z) = z^{-1}$.

Most transfer functions are more complicated than that of Figure 8.1 and can include polynomials of z in both the numerator and denominator, just as analog transfer functions contain polynomials of s in the numerator and denominator:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2} + \ldots + b[K]z^{-K}}{1 + a[1]z^{-1} + a[2]z^{-2} + \ldots + a[L]z^{-L}} \tag{8.12}$$

where the $b[k]$'s are constant coefficients of the numerator and the $a[l]$'s are coefficients of the denominator.[3] Although $H(z)$ has a structure similar to the Laplace domain transfer function $H(s)$, there is no simple relationship between them.[4] For example, unlike analog systems, the order of the numerator, K, need not be less than, or equal to, the order of the denominator, L, for stability. In fact, systems that have a denominator order of 1 (i.e., such as FIR filters) are more stable than those that have higher-order denominators. Remember that in the time domain, the powers of z indicate the position of the sample in a data sequence.

But just as in the Laplace transform, an LTID system is completely defined by the denominator, a, and numerator, b, coefficients. Equation 8.12 can be more succinctly written as:

$$H(z) = \frac{\sum_{k=0}^{K} b[k]z^{-k}}{\sum_{l=0}^{L} a[l]z^{-l}} \tag{8.13}$$

where $a[0] = 1.0$. From the digital transfer function, $H(z)$, it is possible to determine the output given any input:

$$Y(z) = X(z)H(z), \quad y(n) = Y^{-1}(z) \tag{8.14}$$

where $Y^{-1}(z)$ is the inverse z-transform. Just as with the Laplace transform, z-transform tables can be used to convert from the (discrete) time domain to the z-domain. Extensive z-transform tables can easily be found on the web.

The z-transform transfer function (Equations 8.13) can be used in the same manner as the Laplace transfer function to find the output of an LTID system to a given input. Multiply the transfer function by the z-transform of the input and rearrange the resulting equation to match an entry in the z-transform table. You may have to go through a few of the same algebraic manipulations such as partial fraction expansion, but the basic idea is the same.

[3]This equation can be found in a number of different formats. A few authors reverse the role of the a and b coefficients with a's being the numerator coefficients and b's being the denominator coefficients. Another variation is to reverse the coefficient order, with $b[0]$ representing the highest power of z. Sometimes, the denominator coefficients are written with negative signs changing the nominal sign of the a coefficients. We will follow the format used by MATLAB with the exception that coefficient series begins with $b[0]$ and $a[1]$ so the coefficient index is the same as the power of z. Since MATLAB does not allow an index of 0, this equation is formatted with the series index beginning with `b[1]`and `a[2]` (so `a[1]` is always 1.0 in MATLAB).

[4]Nonetheless, the z-transfer function borrows from the Laplace terminology, so the term "pole" is sometimes used for denominator coefficients and the term "zeros" for numerator coefficients.

The inverse z-transform method is tedious, not amenable to computer analysis (except possibly for finding the roots of a polynomial), and, fortunately, not necessary for filter analysis. With digital filters, we are interested only in the filter's spectrum. When we determine the spectrum of a system, any system, if we assume steady-state conditions. This means we can avoid solving for the inverse z-transform. We simply substitute $e^{j\omega}$ for z in the transfer function and evaluate the result using the Fourier transform as shown in the next section.[5]

8.2.3 Transformation From the z-Domain to the Frequency Domain

To transform a z-transform to the (discrete) frequency domain, we substitute $e^{j\omega}$ for z. This is analogous to substituting $e^{j\omega}$ for s to convert a Laplace transfer function to the frequency domain. With this substitution, Equation 8.13 becomes:

$$H(m) = \frac{\sum_{k=0}^{K-1} b[k]e^{-j\omega k}}{\sum_{l=0}^{L-1} a[l]e^{-j\omega l}} = \frac{\sum_{k=0}^{K-1} b[k]e^{\frac{-j2\pi mk}{K}}}{\sum_{l=0}^{L-1} a[l]e^{\frac{-j2\pi ml}{L}}} = \frac{FT\{b[k]\}}{FT\{a[l]\}} \tag{8.15}$$

where K is the number of b (numerator) coefficients and L is the number of a (denominator) coefficients. As in all Fourier transforms, the actual frequency can be obtained from the variable m by multiplying m by f_s/N or, equivalently, $1/(NT_s)$. In practice, both the numerator and denominator coefficients are zero padded to be the same length, a length large enough to generate a smooth spectrum. Equation 8.15 is easily implemented in MATLAB as shown in Example 8.1.

EXAMPLE 8.1

Find and plot the frequency spectrum (magnitude and phase) of a system having the following digital transfer function. Assume this system is used with signals that are sampled at 1000 Hz.

$$H(z) = \frac{0.2 + 0.5z^{-1}}{1 - 0.2z^{-1} + 0.8z^{-2}} \tag{8.16}$$

Solution: First identify the a and b coefficients from the digital transfer function. From Equation 8.16, the numerator coefficients are $b = [0.2, 0.5]$ and the denominator coefficients are $a = [1.0, -0.2, 0.8]$. Then solve Equation 8.15 using these coefficients. Zero pad both coefficients to the same large number of samples to get a smooth spectrum. (Here we use $N = 512$, which is more than sufficient.) When plotting, use a frequency vector: $f = mf_s/N$.

```
% Example 8.1 Plot the Frequency characteristics of an LTID system defined
% by the z-transfer function, Equation 8.16.

%
fs = 1000;                % Assumed sampling frequency
```

[5]For an extensive discussion of the inverse z-transform, see, for example, Lathi, 2005.

```
N = 512;                              % Number of points (arbitrary)
%
%  Define a and b coefficients based digital transfer function
a = [1 -.2 .8];                       % Denominator coefficients
b = [.2 .5];                          % Numerator coefficients
%

H = fft(b,N)./fft(a,N);               % Equation 8.15 with zero padding

Hm = 20*log10(abs(H));                % Get magnitude in dB
Theta = (angle(H))*360/(2*pi);        % and phase in deg.
f = (1:N/2) *fs/N;                    % Frequency vector for plotting
  .......plot magnitude and phase transfer function with grid and labels.......
```

Result: The spectrum of the system defined in Equation 8.16 is presented in Figure 8.2. The spectrum has a hump at around 240 Hz, indicating that the system is underdamped with an undamped resonant frequency (ω_n) of approximately 240 Hz.

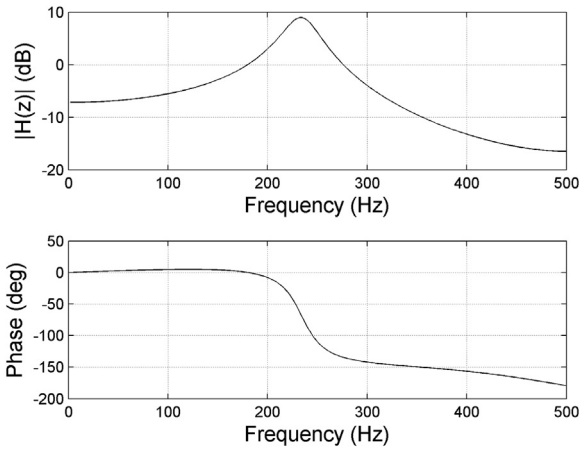

FIGURE 8.2 Plot of the frequency characteristic (magnitude and phase) of the digital transfer function given in Equation 8.16. The spectrum describes an underdamped system with a resonant frequency of approximately 240 Hz.

This example demonstrates how easy it is to determine the time and frequency behavior of a system defined by a z-transform transfer function. The approach used here can readily be extended to any transfer function of any complexity. Other examples of the use of the z-transform are given in the problems.

The next question you might ask is, how do we implement a digital filter, or any digital system, given its z-transfer function? That is, how do we apply the digital filter to a digital signal? Again we would like to avoid having to take the inverse z-transform and implement the system on a computer. For this, we turn to the time domain representation of digital systems, a representation that uses "difference equations."

8.3 DIFFERENCE EQUATIONS

Difference equations are the discrete-time equivalent of continuous differential equations. A simple first-order differential equation and its discrete-time counterpart illustrate this relationship. Given a typical first-order differential equation:

$$\frac{dy}{dt} + ky(t) = x(t) \tag{8.17}$$

If $x(t)$ is converted to a discrete signal by sampling at T_s seconds, then Equation 8.17 becomes:

$$\lim_{T_s \to 0} \frac{y[n] - y[n-1]}{T_s} + ky[n] = x[n]$$

Assuming T_s is small, but not 0, multiplying through by T_s and combining $y[n]$'s yields:

$$(1 + kT_s)y[n] - y[n-1] = x[n]T_s \tag{8.18}$$

Normally the coefficient of $y[n]$ is normalized to 1:

$$y[n] - \left(\frac{1}{1+kT_s}\right)y[n-1] = \left(\frac{T_s}{1+kT_s}\right)x[n] \tag{8.19}$$

Equation 8.19 is a difference equation, so-called because it is based on the difference between samples (i.e., $y[n]$ and $y[n-1]$). Equation 8.19 is the discrete-time equivalent of the continuous differential equation given in Equation 8.17. The highest order difference in either x or y indicates the order of the difference equation and corresponds to the order of the equivalent differential equation. In Equation 8.18, the highest difference is 1.0 (between the n and $n-1$ terms), so this is a first-order difference equation. This is expected since it is the digital version of a first-order differential equation, Equation 8.17. A second-order difference equation would have an $n-2$ term (in either x or y).

Although the input–output relationship can be determined from equations in the form of Equation 8.18 or 8.19, it is more common to use difference equations that are arranged to be discrete versions of convolution. To convert from the z-domain transfer function of Equation 8.13, we begin with the expanded version of Equation 8.13 given in Equation 8.12 and repeated here:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{Z[y[n]]}{Z[x[n]]} = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2} + \ldots + b[K]z^{-K}}{1 + a[1]z^{-1} + a[2]z^{-2} + \ldots + a[L]z^{-L}}$$

Clearing the fraction in Equation 8.12 by multiplying both sides by both denominators gives:

$$\begin{aligned} &Z[y[n]]1 + Z[y[n]]a[1]z^{-1} + Z[y[n]]a[2]z^{-2} + \ldots + Z[y[n]]a[L]z^{-L} = \\ &Z[x[n]]b[0] + Z[x[n]]b[1]z^{-1} + Z[x[n]]b[2]z^{-2} + \ldots + Z[x[n]]b[K]z^{-K} \end{aligned} \tag{8.20}$$

Now we apply the time shift interpretation of z^{-k} given in Equation 8.11 and repeated here:

$$z^{-k} \Leftrightarrow x[n-k] \quad (\text{or } z^{-k} \Leftrightarrow y[n-k])$$

Taking the inverse z-transform of each term in Equation 8.20 using this time shift interpretation:

$$y[n] + y[n-1]b[1] + y[n-2]b[2] + \ldots + y[n-L]b[L] = \\ x[n]b[0] + x[n-1]b[1] + x[n-2]b[2] + \ldots + x[n-K]b[K]$$

Note that $a[0]$ is assumed to be 1.0 (Equation 8.13) so $y[n]a[0] = y[n]$. Solving for $y[n]$ in terms of $x[n]$ and past values of $y[n]$:

$$\begin{aligned} y[n] = {} & x[n]b[0] + x[n-1]b[1] + x[n-2]b[2] + \ldots + x[n-K]b[K] \\ & -y[n-1]b[1] - y[n-2]b[2] - \ldots - y[n-L]b[L] \end{aligned}$$

Combining the sums:

$$y[n] = \sum_{k=0}^{K-1} b[k]x[n-k] - \sum_{l=1}^{L-1} a[l]y[n-l] \tag{8.21}$$

This difference equation defines any general LTID system in the discrete-time domain. It contains two summations: a summation involving the b coefficients applied to the input signal and a summation of the a coefficients applied to delayed samples of the output signal. Note that the summation of a coefficients begins at $l = 1$ (since $a[0] = 1$), so only past values of the output signal, y, contribute to the current output signal. Nonetheless, the a coefficients >1 are former outputs and they contribute, in part, to the current output ($y[n]$), so they are called "recursive coefficients."

The difference equation (Equation 8.21) also provides us with a method to find the output, $y[n]$, to any input, $x[n]$, without having to deal with the inverse z-transform. Compare the two terms in Equation 8.21 with the convolution equation introduced in Chapter 5 (Equation 5.3) and repeated here:

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{8.22}$$

The two summations in Equation 8.21 have the same form as the convolutions equation. In fact, if $h[k] = b[k]$ the first summation of Equation 8.21 is identical to the convolution equation, Equation 8.22. So the output to any discrete system can be obtained by a double convolution, convolving the input with the b coefficients and past values of the output with the a coefficients. Implementation of Equation 8.21 in MATLAB is straightforward as shown in the next example.

EXAMPLE 8.2

Find the output of the system defined by Equation 8.16 in Example 8.1 to a unit step input. Use the same values for f_s and make the signal length 256 samples.

Solution: Define the `a` and `b` coefficients as vectors then generate the unit step. The first summation in Equation 8.21 can be found by direct convolution of `b` with `x` using MATLAB's `conv` routine. Use the `'same'` option with `x` as the first parameter so the output signal is the same length as the input signal `x`.

Evaluating the second term is a bit more complicated, as it is recursive: the output depends on past values of itself. We must write our own code for this convolution. Since there are three a coefficients, we need to pad the output with two zeros and the first output term. Since $a[0] = 1$, the first output term is just the first term of the convolution of `x` and `b`. We then use a loop to carry out the recursive convolution of the second summation. After the convolution is completed, we left shift the output one position to account for the leading zero padding. The result is plotted against a time vector based on f_s. Only the first 60 points are plotted to emphasize the transient response.

```
% Example 8.2 Compute the step response of the system defined by Equation 8.21
fs = 1000;                    % Sampling frequency (from Example 8.1)
N = 256;                      % Number of points (arbitrary)
%  Define a and b coefficients (from Example 8.1)
a = [1 -.2 0.8];
b = [.2 .5];
% Compute the Step Response
x = [0, ones(1,N-1)];         % Generate a step function
b_conv = conv(x,b,'same');    % Generate first convolution
y = [0 0 b_conv(1)];          % Initialize y with padding and first term
for k = 1:N
  y(k+2) = b_conv(k) - (y(k+1)*a(2) + y(k)*a(3));  % Equation 8.21, second conv
end
y = y(2:N+1);                 % Shift y to account for padding
t = (1:N)/fs;                 % Time vector for plotting
  ..... plot and label.......
```

Results: The time response generated by this program is shown in Figure 8.3. The step response is clearly that of an underdamped system showing a decaying oscillation. We can estimate the time difference between oscillatory peaks as roughly 0.004 s, which is equivalent to 250 Hz. This is close to the resonant peak shown in Figure 8.2 estimated to be approximately 240 Hz.[6] From the time plot we could estimate the damping factor of this system, but that is easier to do using the spectrum in Figure 8.2.

Equation 8.21 is very useful for implement digital filters. Once we know the appropriate filter coefficients, any signal could be filtered using the code in Example 8.2. However, this is unnecessary because Equation 8.21 is so useful that MATLAB has a routine that implements this equation with one statement:

```
y = filter(b,a,x);            % Find the output of a discrete system
```

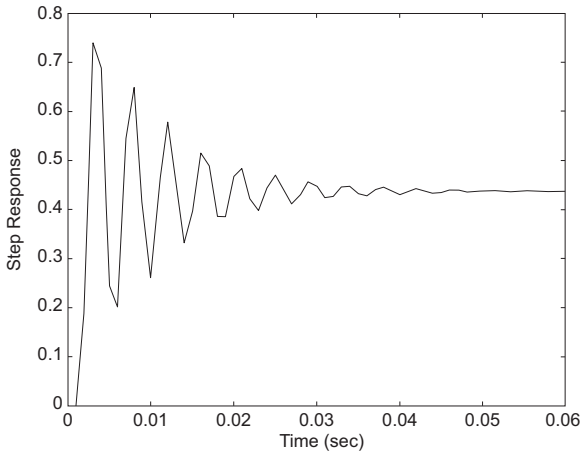

FIGURE 8.3 The step response of the discrete system defined by the transfer function in Equation 8.16. The spectrum of this system was determined in Example 8.1 and shown in Figure 8.2. The step response is clearly that of an underdamped system as is expected from its spectrum. The oscillatory peaks are spaced about 0.004 s apart, suggesting a damped resonance around 240 Hz in rough agreement with the spectrum in Figure 8.2.

where `b` is a vector of b coefficients, `a` is a vector of a coefficients, and `x` is the input signal. The `filter` routine is much faster than the code in Example 8.2. In a test run with more samples, the code in Example 8.2 required 6.316 ms, whereas `filter` required 0.309 ms, more than 20 times faster. It is hard to outperform MATLAB.

[6]The resonant oscillations in Figure 8.3 reflect the damped resonant frequency ω_d, whereas the spectral peak is at the undamped natural frequency, ω_n. As given by Equation 7.34, the damped frequency is slightly less than the undamped natural frequency: $\omega_d = \omega_n\sqrt{1-\delta^2}$. But in this system δ is quite small: approximately 0.07 based on the height of the resonance peak in Figure 8.2. So ω_d should be very close to ω_n. More than likely, the difference (222 vs. 240 Hz) is due to the difficulty accurately reading the time and spectral plots.

We can now find the output of any LTID system given its transfer function, or only the coefficients of that transfer function. To apply a digital filter to any waveform, we use Equation 8.21 embodied in MATLAB as `filter (b,a,x)`. But we still need to find the a and b coefficients that give the filtering we desire. In other words, we need to find coefficients that define a system having a spectrum that attenuates the noise in our signal, but not the frequencies of interest. Finding these coefficients is called "filter design" or "filter synthesis", and is the subject of the remainder of this chapter.

8.4 LINEAR FILTERS—INTRODUCTION

Filtering is a process that alters a signal in some desired manner and is usually described in terms of its spectrum. Filters exist in both the analog or digital domain. The former consists of analog electronics such as those described in Chapter 15. Because of the problem of aliasing due to sampling (see Section 4.1.1), most biomeasurement systems have an analog low-pass

filter, an "anti-aliasing filter," that is applied to the signal before it is digitized. Once the signal is digitized, additional filtering can be done using digital filters.

Digital filters can be adaptive, modifying their coefficients to respond to certain features of the signal, or they can be fixed with constant filter coefficients. Here we concentrate on fixed filters implemented by discrete linear (LTID) systems.

While we implement filters in the time domain using Equation 8.21, we think about filters in the frequency domain; specifically, how a filter reshapes the signal's spectrum. A frequent goal of filtering is to reduce noise. Most noise occurs over a wide range of frequencies (noise is broadband), whereas most signals have a limited frequency range (signals are narrowband or bandlimited). Reducing the energy of frequencies outside the signal range reduces noise, but not signal; the signal to noise ratio (SNR) is improved. Such filtering is usually done using a fixed digital filter: a specially designed LTID system that reshapes a signal's spectrum in some well-defined, and we hope, beneficial manner.

Chapter 5 touched on the use of simple filters. Example 5.6 used a three-point running average, implemented through convolution, to reduce noise in an electrocardiogram (ECG) signal (Figure 5.18). The spectrum of this system was found using the Fourier transform to be that of a low-pass filter (Figure 5.19). Putting the three-point average in context of what we have learned thus far, it was a digital filter with three b coefficients of 0.33 each and had no a coefficients except $a[0] = 1$. The z-transform transfer function of this filter would be:

$$H(z) = \frac{0.33 + 0.33z^{-1} + 0.33z^{-2}}{1} \tag{8.23}$$

Discrete systems for filtering can be classified into two groups depending on their impulse response characteristics: those that have short well-defined impulse responses and those that produce impulse responses that theoretically go on forever. The former are termed finite impulse response ("FIR") filters and the latter infinite impulse response ("IIR") filters.

FIR filters have only one a coefficient, $a[0] = 1$, but any number of b coefficients. Since they do not have additional a coefficients, the second, recursive summation in Equation 8.21 does not occur, so FIR filters are nonrecursive. As shown in Chapter 5, their spectrum can be found by taking the Fourier transform of the b coefficients. This is in agreement with Equation 8.15 because the Fourier transform of a single a coefficient with a value of 1.0, would be simply 1.0, making the denominator 1.0 and the numerator FT$\{b[k]\}$. The three-point moving average defined by the transfer function in Equation 8.23 is an example of an FIR filter. FIR filters can be implemented using convolution or the `filter` routine where `a` = 1.

IIR filters have both a and b coefficients. They include the recursive operation defined by the second term in Equation 8.21. They are implemented by the `filter` routine, which, again, embodies Equation 8.21. Real-world IIR filters do not really have infinite impulse responses, but they do have longer impulse responses than FIR filters having the same number of total coefficients. Figure 8.4 shows the impulse response of an IIR filter with 5 a coefficients and 5 b coefficients and an FIR filter with 10 b coefficients. As shown in Figure 8.4, the IIR filter has an impulse response that is longer than the FIR filter with the same number of total coefficients. Both filter types are easy to implement in basic MATLAB; however, the design of IIR filters is best done using the MATLAB Signal Processing Toolbox. The advantages and disadvantages of each of the filter types are summarized in the next section.

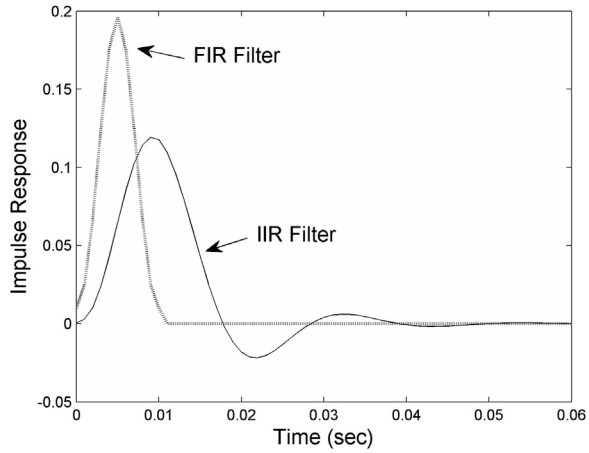

FIGURE 8.4 The impulse response of infinite impulse response (IIR) and finite impulse response (FIR) filters having the same number of total coefficients. The IIR filter has 5 *a* coefficient and 5 *b* coefficients, whereas the FIR filter has 10 *b* coefficients. The impulse response of the IIR filter is longer.

8.4.1 Filter Properties

Although you might think that a filter could have any imaginable magnitude and phase curve, in practice the variety of spectral characteristics of most filters is limited. Filter spectra can be defined by three basic properties: basic filter type, bandwidth, and attenuation slope.

8.4.1.1 Filter Bandwidth

Bandwidth was defined in Section 4.5 in terms of signals, but the same definitions apply to systems. The magnitude spectra of an ideal and a real filter are shown in Figure 4.21 (Chapter 4). The cutoff frequency of a filter is taken where the spectrum has decreased by 3 dB from its unattenuated level, usually the level in the passband (see Figure 4.21B–D).

For a filter, the nominal gain region, the frequency region where the signal is not attenuated more than 3 dB, is termed the "passband," and the gain of the filter in this region is called the "passband gain." The passband gain of all filters in Figure 4.21 is 1.0. The frequency region where the signal is attenuated more than 3 dB is termed the "stopband."

8.4.1.2 Filter Type

Filters are usually classed based on the range of frequencies they do not suppress, that is, the frequencies they pass. A "low-pass" filter allows low frequencies to pass with minimum attenuation, whereas higher frequencies are attenuated. Conversely, a "high-pass" filter permits high frequencies to pass, but attenuates low frequencies. "Band-pass" filters allow a range of frequencies through, whereas frequencies above and below this range are attenuated. An exception to this terminology is the "band-stop" filter, which passes frequencies on either side of a range of attenuated frequencies: it is the inverse of the band-pass filter. The frequency characteristics of the four filter types are shown in Figure 8.5.

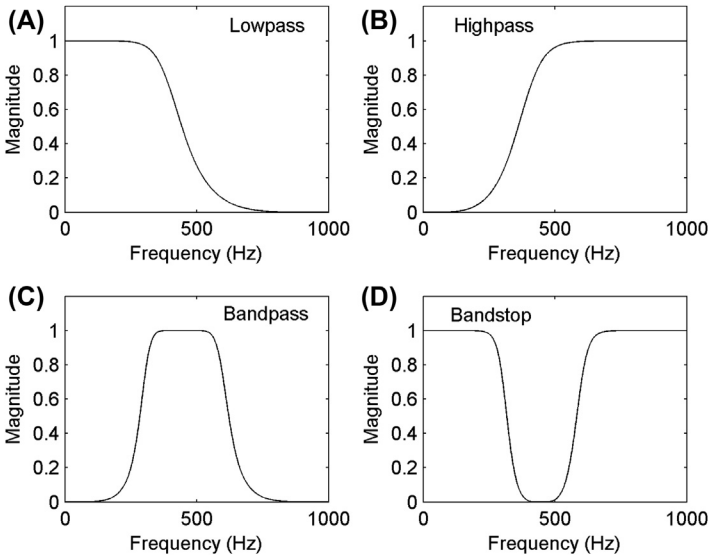

FIGURE 8.5 Magnitude spectral curves of the four basic filter types. (A) Low pass $f_c = 400$ Hz; (B) high pass $f_c = 400$ Hz; (C) band pass $f_{c1} = 300$ Hz, $f_{c2} = 600$ Hz; and (D) band-stop $f_{c1} = 300$ Hz, $f_{c2} = 600$ Hz.

8.4.1.3 Filter Attenuation Slope—Filter Order

Filters are also defined by the sharpness with which they increase or decrease attenuation as frequency varies. The attenuation slope is sometimes referred to as the filter's "rolloff." Spectral sharpness is specified in two ways: as an initial sharpness in the region where attenuation first begins and as a slope further along the attenuation curve.

For FIR filters, the attenuation slope is directly related to the length of the impulse response, i.e., the number of b coefficients. In FIR filters, the filter order is defined as one less than the number of b coefficients. Figure 8.6 shows the spectral characteristics of four FIR filters that have the same cutoff frequency but increasing filter orders, and the commersurate increase in attenuation slope is apparent.

The attenuation slope of IIR filters follows the pattern found for transfer functions analyzed in Chapter 6. The magnitude of the slope is determined by the order of the denominator polynomial: a first-order system has a slope of 20 dB/decade, whereas a second-order system has a slope of 40 dB/decade. This generalizes to higher-order IIR filters so that an nth-order IIR filter has an attenuation slope of $20n$ dB/decade. In Chapter 15, we examine analog circuits and find that the order of a circuit's denominator polynomial is related to the complexity of the electrical circuit, specifically the number of energy storage elements in the circuit.

8.4.1.4 Filter Initial Sharpness

For both FIR and IIR filters, the attenuation slope increases with the filter order, although for FIR filters, it does not follow the neat 20 dB/decade per filter order slope of IIR filters. For IIR filters, it is possible to increase the initial sharpness of the filter's attenuation characteristics without increasing the order of the filter, if you are willing to accept some unevenness, or

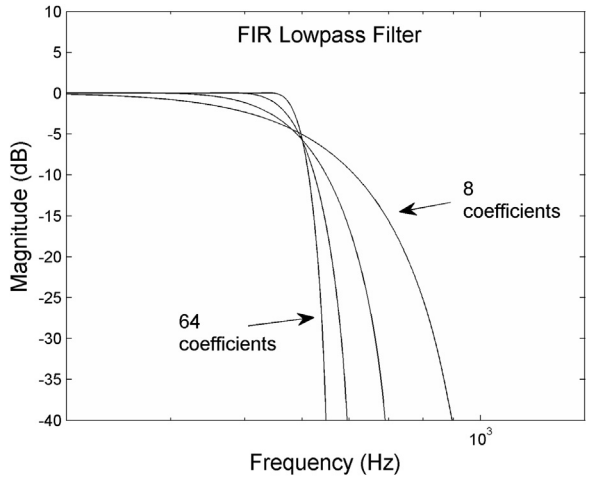

FIGURE 8.6 The magnitude spectral characteristics of four finite impulse response filters that have the same cutoff frequency, $f_c = 500$ Hz, but increasing filter orders of 8, 16, 32, and 64. The higher the filter order, the steeper the attenuation slope.

"ripple," in the passband. Figure 8.7 shows two low-pass, fourth-order IIR filters, differing in the initial sharpness of the attenuation. The one marked Butterworth has a smooth passband, but the initial attenuation is not as sharp as the one marked Chebyshev, which has a passband that contains ripples. This figure also shows why the Butterworth filter is popular in biomedical engineering applications: it has the greatest initial sharpness while maintaining a smooth passband. For most biomedical engineering applications, a smooth passband is needed.

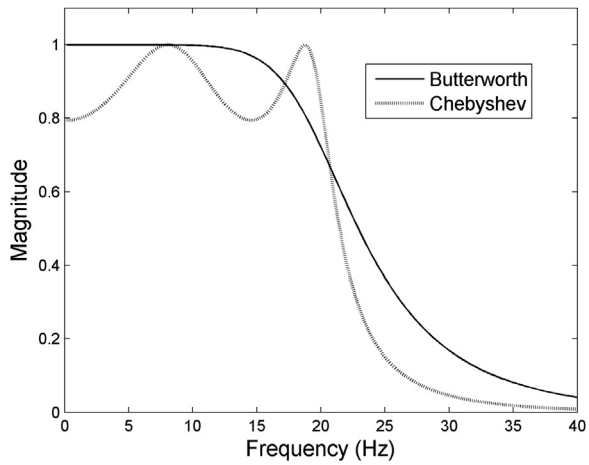

FIGURE 8.7 Two fourth-order infinite impulse response filters with the same cutoff frequency of 20 Hz, but differing in the sharpness of the initial slope. The filter marked "Chebyshev" has a much steeper initial slope, but contains ripple in the passband. The final slope also looks steeper, but would not be if it were plotted on a log scale.

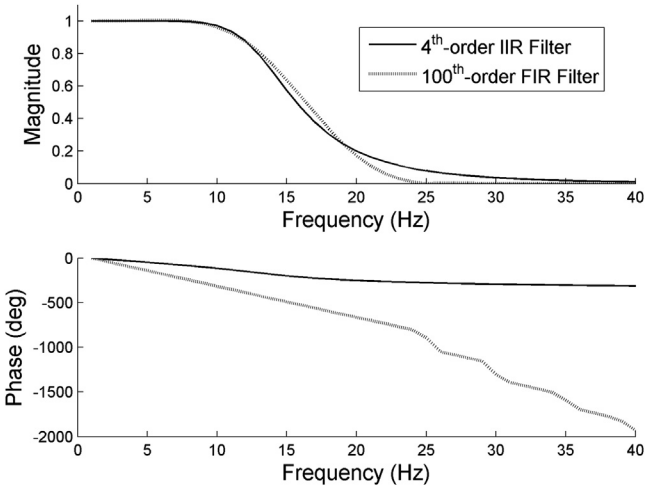

FIGURE 8.8 Comparison of the magnitude and phase spectra of a 100th-order finite impulse response (FIR) filter and a 4th-order infinite impulse response (IIR) filter, both having a cutoff frequency, f_c, of 14 Hz. (Upper graph) The magnitude spectra show approximately the same attenuation slope, but the FIR filter requires 10 times the number of coefficients (a fourth-order IIR filter has 10 coefficients, including $a[0] = 1.0$). (Lower graph) The phase spectra show that the FIR filter has a greater phase change, but that change is linear within the passband. This turns out to be an important feature of FIR filters.

8.4.2 Finite Impulse Response Versus Infinite Impulse Response Filter Characteristics

There are several important behavioral differences between FIR and IIR filters aside from the characteristics of their impulse responses. FIR filters require more coefficients to achieve the same filter slope. Figure 8.8A compares the magnitude spectra of a 4th-order IIR filter with a 100th-order FIR filter having the same cutoff frequency of 14 Hz. They both have approximately the same slope, but the FIR filter requires more than 10 times the number of coefficients. (A 4th-order IIR filter has 10 coefficients and a 100th-order IIR filter has 125 coefficients). In general, FIR filters do not perform as well for low cutoff frequencies as is the case here, and the discrepancy in filter coefficients is less if the cutoff frequency is increased. This behavior is explored in the problem set. Another difference is the phase characteristics as shown for the two filters in Figure 8.8B. The FIR has a larger change in phase, but the phase change is linear within the passband region of 0−14 Hz.

Figure 8.9 shows the application of the two filters to a fetal ECG signal that contains noise. Both low-pass filters do a good job of reducing the high-frequency noise, but, again, the FIR filter has 10 times the number of coefficients, and the computing time required to apply this filter is correspondingly longer. (For a 10,000 sample signal, the IIR filter required 0.56 ms, whereas the FIR filter required 7.04 ms on a typical personal computer.) The filtered data are also shifted to the right in comparison with the original signal, more so for the FIR filtered data. This is because both filters are causal (see Section 1.4.3) and the output of these filters depends on past values of the input. Causal and noncausal filtering are discussed in the next section (Section 8.4.3).

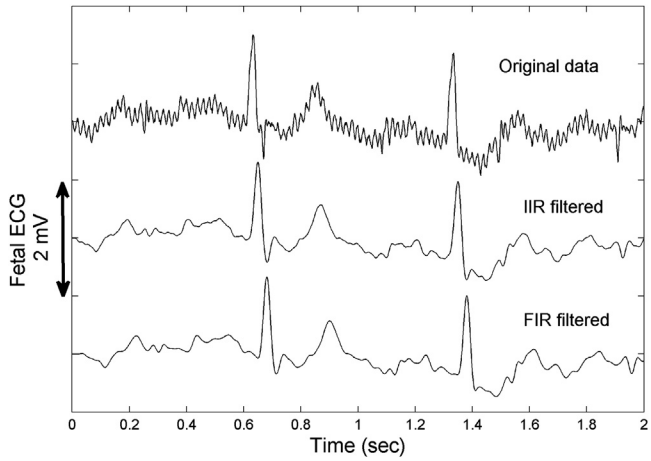

FIGURE 8.9 A fetal electrocardiography signal recorded from the abdomen of the mother. The unfiltered (upper) signal is filtered by the infinite impulse response (middle) and finite impulse response (FIR) (lower) filters whose spectra are shown in Figure 8.8. The 60-Hz noise in the original is considerably reduced in both filtered signals, but, again, the FIR filter requires 10 times the data samples and a proportionally longer computation time. Both filters induce a time shift because they are causal filters as described later.

The linear phase property of FIR filters shown in Figure 8.8 makes FIR filters much easier to design since we need only be concerned with the magnitude spectrum of the desired filter. FIR filters are also more stable than IIR filters. Since IIR filters are recursive, they can produce unstable oscillatory outputs for certain combinations of filter coefficients. This is not a problem if the coefficients are fixed as in standard filters, but in adaptive filters, where filter coefficients are modified on the fly, severe problems can occur. For this reason, adaptive filters employ FIR filters. IIR filters are also unsuitable for filtering two-dimensional data, so FIR filters are used exclusively for filtering images. A summary of the benefits and appropriate applications of the two filter types is given in Table 8.2.

TABLE 8.2 Finite Impulse Response (FIR) Versus Infinite Impulse Response (IIR) Filters: Features and Applications

Filter Type	Features	Applications
FIR	Easy to design Stable Applicable to two-dimensional data (i.e., images)	Fixed, one-dimensional filters Adaptive filters Image filtering
IIR	Require fewer coefficients for the same attenuation slope, but can become unstable Particularly good for low cutoff frequencies. Mimic analog (circuit) filters	Fixed, one-dimensional filters, particularly at low cutoff frequencies Real-time applications where speed is important

8.4.3 Causal and Noncausal Filters

If a filter uses only the current and past data samples, it is a "causal" filter. As noted previously, all real-world systems must be causal since they do not have access to the future; they have no choice but to operate on current and past values of a signal. However, if the data are already stored in a computer, it is possible to use future signal values along with current and past values to compute an output signal, that is, future data with respect to any given data sample. Filters (or systems) that use future values of a signal in their computation are noncausal.

The motivation for using future values in filter calculations is provided in Figure 8.10. The upper curve in Figure 8.10A is the response of the eyes to a target that jumps inward in a step-like manner. (The curve is actually the difference in the angle of the two eyes with respect to the straight-ahead position.) These eye movement responses are corrupted by 60-Hz noise riding on top of the signal, a ubiquitous problem in the acquisition of biological signals. A simple FIR filter consisting of 10 equal coefficients of 0.1 (i.e., a 10-point moving average) was applied to the noisy data and this filter does a good job of removing the noise as shown in the lower two curves. The filter was applied in two ways: as a causal filter using only current and past values of the eye movement data, (output shown in lower curve of Figure 8.7A), and as a noncausal filter using an equal number of past and future values, (output shown in

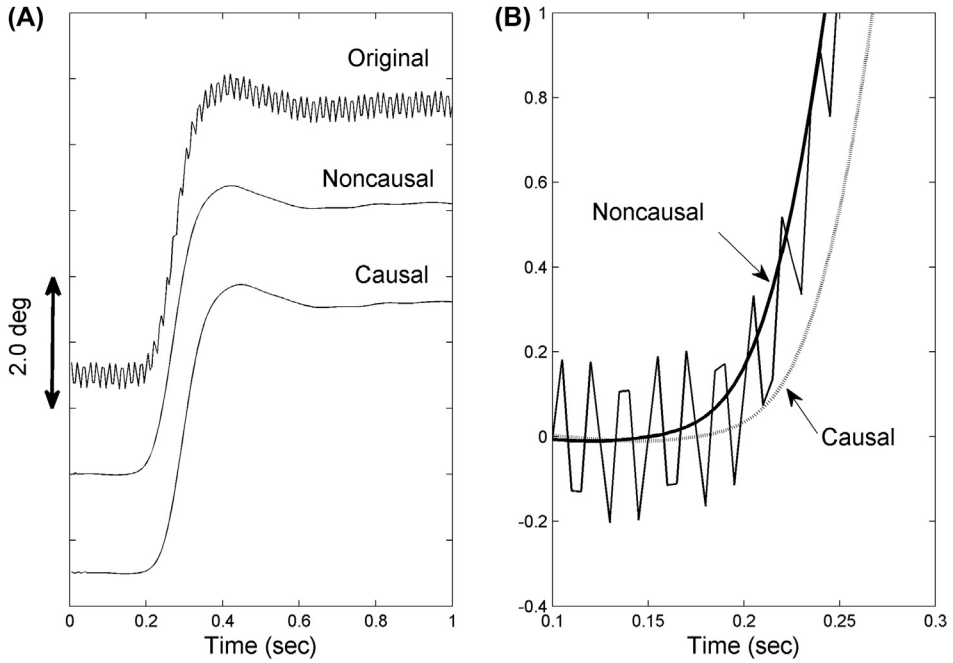

FIGURE 8.10 (A) Eye movement data containing 60-Hz noise and the same response filtered with a 10-point moving average filter applied in a causal and noncausal mode. (B) Detail of the initial response of the eye movement showing the causal and noncausal filtered data superimposed. The noncausal filter overlays the original data, whereas the causal filter produces a time shift in the filtered response.

middle curve of Figure 8.7A). The causal filter was implemented using MATLAB's `conv` routine, and so was the noncausal filter, but with the option `'same'` i.e.:

```
y = conv(x,b,'same');      % Noncausal filtering
```

where `x` is the signal and `b` is the filter coefficients. When the `'same'` option is invoked, the convolution algorithm returns only the center section of output, effectively shifting future values into the present. In Chapter 5, Figure 5.17 shows the noncausal implementation of a three-point moving average, as one of the values in the average is taken from a sample ahead (i.e., in the future) of the output sample.

Both filters do a good job of reducing the 60-cycle noise, but the causal filter has a slight delay. In Figure 8.7B, the initial responses of the two filters are plotted superimposed over the original data and the delay in the causal filter is apparent. Eliminating the delay or time shift inherent in causal filters is the primary motivation for using noncausal filters. In Figure 8.10, the time shift in the output is small, but can be much larger if filters with longer impulse responses are used or if the data are passed through multiple filters. However, in many applications a time shift does not matter, so causal filter implementation is adequate.

8.5 DESIGN OF FINITE IMPULSE RESPONSE FILTERS

For FIR filters, the impulse response is the filter coefficients, $b[k]$. Since there are no a coefficients except $a[0]$, FIR filters are implemented using convolution and in MATLAB, either the `conv` or `filter` routine can be used. Referring to Equation 8.15, since the Fourier transform of the a coefficient is 1.0, the FIR filter spectrum is just the Fourier transform of the b coefficients.

To design an FIR filter, we start with the desired spectrum. Since the desired spectrum is the Fourier transform of the b coefficients, to find these coefficients, all we need do is take the inverse Fourier transform of that spectrum. We need only the magnitude spectrum because FIR filters have a linear phase, so the phase is uniquely determined by the magnitude spectrum. Occasionally we might want some specialized frequency characteristic, but usually we just want to separate out a selected frequency range from everything else, for example, the signal frequency range from everything else. Also, we usually want that separation to be as sharp as possible. In other words, we usually prefer an ideal filter such as that shown in Figure 4.21A with a particular cutoff frequency. The spectrum of an ideal low-pass filter has a shape like a rectangular window, so sometimes the filters are called "rectangular window" filters.[7]

This spectrum of an ideal filter is a fairly simple function, so we ought to be able to find the inverse Fourier transform analytically from the defining equation given in Chapter 3 (a rare situation in which we are not totally dependent on MATLAB). When using the complex form, we must include the negative frequencies, so the desired filter's frequency characteristic is as

[7]This filter is sometimes just called a "window filter," but the term "rectangular window filter" is used here so as not to confuse such a filter with a "window function" as described in Chapter 4. This can be particularly confusing since, as shown below, rectangular window filters use window functions, but the two words (window filter and window function) mean two very different things!

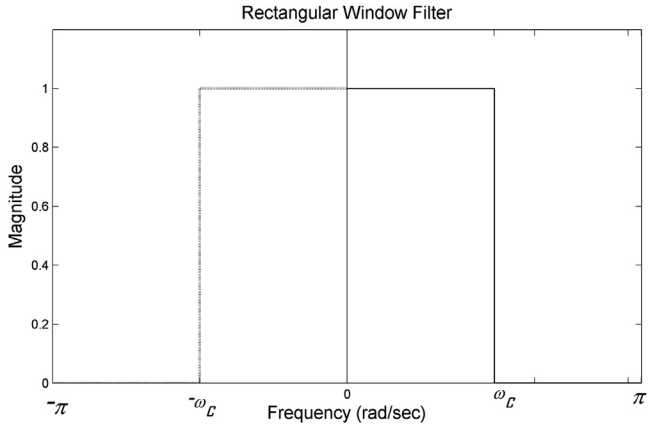

FIGURE 8.11 The magnitude spectrum of a rectangular window filter: an ideal filter. The negative frequency portion of this filter is also shown because it is needed in the computation of the complex inverse Fourier transform. The frequency axis is in radians per second and the cutoff frequency is ω_c.

shown in Figure 8.11. Frequency is shown in radians per seconds to simplify the derivation of the rectangular window impulse response and the cutoff frequency is ω_c.

Since we are trying to solve for the inverse Fourier transform analytically, we use the continuous form given in Equation 3.27, to solve for a continuous series of coefficients $b(t)$, and then convert the result to discrete numbers, $b[k]$.

$$b(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} B(\omega)e^{j\omega t}d\omega = \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 1e^{j\omega t}d\omega \tag{8.24}$$

since the window function is 1.0 between $\pm\omega_c$ and zero elsewhere. Integrating and putting in the limits:

$$b[t] = \frac{1}{2\pi}\frac{e^{j\omega t}}{jt}\Bigg|_{-\omega_c}^{\omega_c} = \frac{1}{\pi t}\frac{e^{j\omega_c t} - e^{-j\omega_c t}}{2j} \tag{8.25}$$

The term $\frac{e^{j\omega_c t} - e^{-j\omega_c t}}{2j}$ is the exponential definition of the sine function and equals $\sin(\omega_c t)$. So the impulse response of a rectangular window is:

$$b(t) = \frac{\sin(\omega_c t)}{\pi t} = \frac{\sin(2\pi f_c t)}{\pi t} \tag{8.26}$$

The impulse response of a rectangular window filter has the general form of a "sinc" function: $\sin(x)/x$. The filter coefficients, $b[k]$, can be obtained from Equation 8.26 by taking discrete values, k, for the continuous time variable t. The resulting coefficients are shown for 64 values of k and two values of f_c (or $\omega_c/2\pi$) in Figure 8.12. These cutoff frequencies are relative to the sampling frequency as explained in the following discussion.

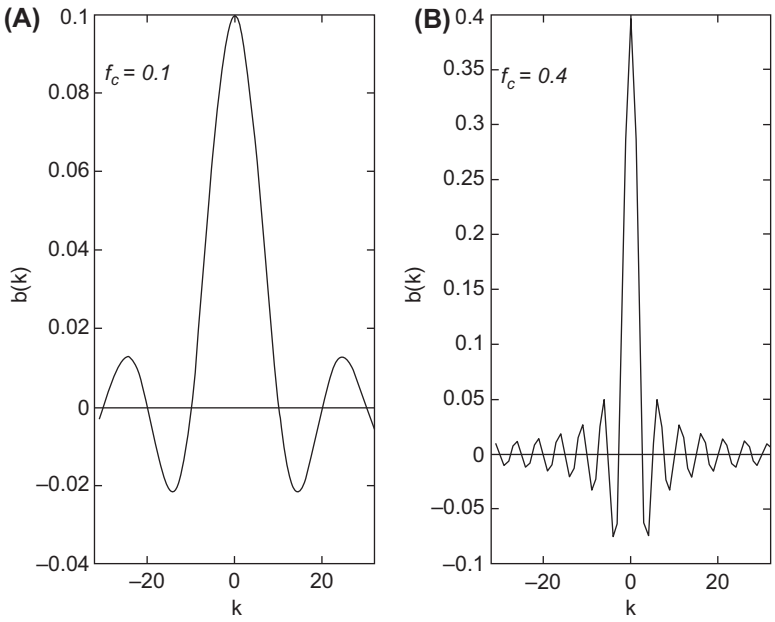

FIGURE 8.12 The impulse response of a rectangular window filter for 64 coefficients determined from Equation 8.26. The cutoff frequencies are given relative to the sampling frequency, f_s. (A) Low-pass filter with a relative cutoff frequency of 0.1 Hz. (B) Low-pass filter with a higher relative cutoff frequency of 0.4 Hz.

From Figure 8.11, the cutoff frequency, ω_c, is relative to a maximum frequency of π. In the spectrum of a discrete signal, the maximum frequency in Hz is the sampling frequency, f_s. So to convert the relative frequencies of Equation 8.26 to the actual frequency, we need merely to multiply f_c by f_s.

$$f_{actual} = f_c f_s \tag{8.27}$$

(Note that MATLAB filter design routines included in the Signal Processing Toolbox use a cutoff frequency relative to $f_s/2$ so $f_{actual} = f_c\,(f_s/2)$.

The symmetrical impulse responses shown in Figure 8.12 have both positive and negative values of k. Since MATLAB requires indexes to be positive, we need to shift the index, k, on the right side of Equation 8.26 to half the total length of the filter. To be compatible with MATLAB, the discrete version of Equation 8.26 becomes:

$$b[k] = \frac{\sin\left(\omega_c\left(k-\frac{L}{2}\right)\right)}{\pi\left(k-\frac{L}{2}\right)} = \frac{\sin\left(2\pi f_c\left(k-\frac{L}{2}\right)\right)}{\pi\left(k-\frac{L}{2}\right)} \tag{8.28}$$

where f_c is the cutoff frequency relative to f_s (Equation 8.27) and L is the length of the filter. If L is odd, then an adjustment is made so that $L/2$ is an integer as shown in the next example.

When Equation 8.28 is implemented, a problem occurs when the denominator goes to zero at $k = L/2$. The actual value of the function for $k = L/2$ can be obtained by applying the limits and noting that $\sin(x) \to x$ as x becomes small:

$$b[L/2] = \lim_{(k-L/2)} \left| \frac{\sin[2\pi f_c(k - L/2)]}{\pi(k - L/2)} \right| = \lim_{(k-L/2)} \left| \frac{2\pi f_c(k - L/2)}{\pi(k - L/2)} \right| = 2f_c \tag{8.29}$$

If frequency is in radians per seconds, the value of $b[L/2]$is:

$$b[L/2] = \lim_{(k-L/2)} \left| \frac{\sin[\omega_c(k - L/2)]}{\pi(k - L/2)} \right| = \lim_{(k-L/2)} \left| \frac{\omega_c(k - L/2)}{\pi(k - L/2)} \right| = \frac{\omega_c}{2} \tag{8.30}$$

There is one serious problem with the logic used thus far: the FIR coefficient equation in Equation 8.26 is infinite. That is, Equation 8.26 (or Equation 8.28) produces nonzero values for all finite values of t or k. Since we are designing filters with a finite number of coefficients, we need to truncate the functions produced by these equations. You might suspect that truncating the filter coefficient limits the filter's performance: we might not get the rectangular cutoff we desire. In fact, truncation has two adverse effects: the filter no longer has an infinitely sharp cutoff and oscillations are produced in the filter's spectrum. These adverse effects are demonstrated in the next example, where we show the spectrum of a rectangular window filter truncated to two different lengths.

EXAMPLE 8.3

Find the magnitude spectrum of the rectangular window filter given by Equation 8.28 for two different coefficient lengths: $L = 17$ and $L = 65$. Use a cutoff frequency of 300 Hz assuming a sampling frequency of 1 kHz (i.e., a relative cutoff frequency of $f_c = 0.3$).

Solution: First generate the filter's impulse response, $b[k]$, by direct implementation of Equation 8.28. Note that the two filter lengths are both odd, so to make the shift a whole number, we reduce L by 1 and then shift by $k - (L-1)/2$. The coefficient $b[L/2]$ should be set to $2f_c$ as given by Equation 8.29. After calculating the coefficients, find the spectrum by taking the Fourier transform of the response. Plot only the magnitude spectrum.

```
%Example 8.3  Generate the coefficients of two rectangular window filters and
%  find and plot their magnitude spectra.
%
N = 256;              % Padding for Fourier transform (arbitrary)
fs = 1000;            % Sampling frequency (assumed)
f = (1:N)*fs/N;       % Frequency vector for plotting
fc = 300/fs;          % Cutoff frequency (normalized to fs)
L = [17 65];          % Filter lengths (filter order + 1)
for m = 1:2           % Loop for the two filter lengths

  for k = 1:L(m)
    n = k-(L-1)/2 ;   % Whole number shift
    if n == 0         % Generate sin(n)/n function.  Use Equation 8.28.
```

```
      b(k) = 2*fc;                    % Case where denominator is zero.
    else
      b(k) = sin(2*pi*fc*n)/(pi*n);   % Filter impulse response
    end
  end
% Now plot the filter's spectrum
  H = fft(b,N);                       % Calculate spectrum
  subplot(1,2,m);                     % Plot magnitude spectrum
  plot(f(1:N/2),abs(H(1:N/2)),'k');
.....labels and title.....
end
```

Results: The spectrum of this filter is shown in Figure 8.13 to have two artifacts associated with finite length. The oscillation in the magnitude spectrum is another example of Gibbs artifact first encountered in Chapter 3 and is due to the truncation of the filter coefficients given by Equation 8.28. In addition, the slope is less steep when the filter's impulse response is shorter.

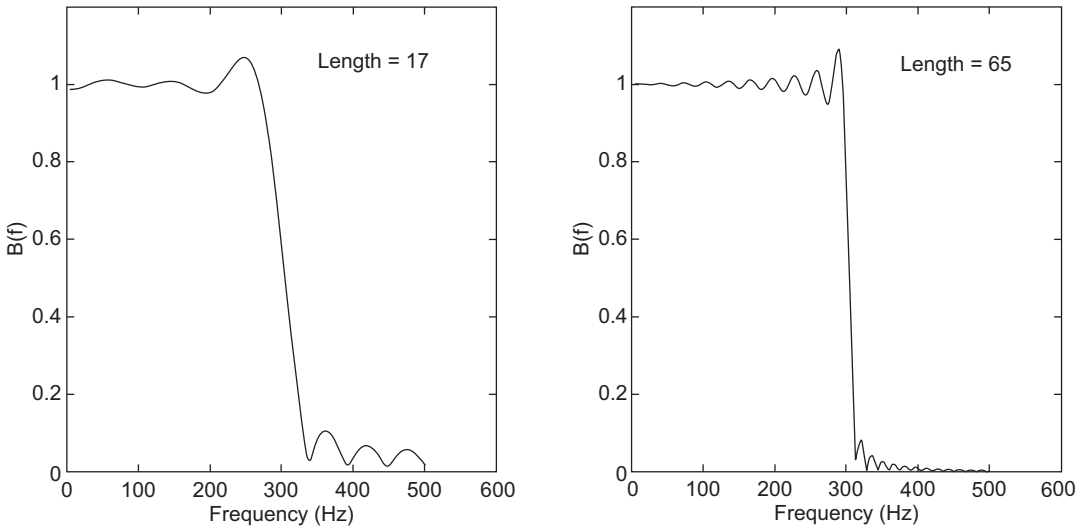

FIGURE 8.13 Magnitude spectrum of two finite impulse response filters based on an impulse derived from Equation 8.28. The impulse responses are abruptly truncated at 17 and 65 coefficients. The low-pass cutoff frequency is 300 Hz for both filters with an assumed sample frequency of 1 kHz. The oscillations seen are Gibbs artifacts and are due to the abrupt truncation of what should be an infinite series. Like the Gibbs artifacts seen in Chapter 3, they do not diminish with increasing filter coefficient length, but do increase in frequency.

We can probably live with the less-than-ideal slope (we should never expect to get an ideal anything in the real world), but the oscillations in the spectrum are serious problems. Since Gibbs artifacts are due to truncation of an infinite function, we might reduce them if the function were tapered toward zero rather than abruptly truncated. In Chapter 4, Section 4.2.4, window functions, such as the Hamming window, are used to improve the spectra obtained

from short data sets. These tapering window functions could be used to reduce the abrupt truncation. There are many different window functions, but the two most popular and most useful for FIR impulse responses are the Hamming and Blackman windows. The Hamming window equation is given in Equation 4.9 and repeated here:

$$w[n] = 0.5 - 0.46\cos\left(\frac{2\pi n}{N}\right) \tag{8.31}$$

where N is the length of the window, which should be the same length as the data. The Blackman window is more complicated, but like the Hamming window, is still easy to program in MATLAB.

$$w[n] = 0.42 - 0.5\cos\left(\frac{2\pi n}{N}\right) + 0.16\cos\left(\frac{4\pi n}{N}\right) \tag{8.32}$$

The following example presents the MATLAB code for the Blackman window[8] and applies it to the filters of Example 8.3.

EXAMPLE 8.4

Apply a Blackman window to the rectangular window filters used in Example 8.3. (Note that the word "window" is used in two completely different contexts in the last sentence as you were warned about in footnote 7.) Calculate and display the magnitude spectrum of the impulse functions after they have been windowed.

Solution: Write a MATLAB function, `blackman(N)`, to generate a Blackman window of length `N`. Apply it to the filter impulse responses using point-by-point multiplication (`.*` operator). Modify the last example by applying the window to the filter's impulse response before taking the Fourier transform.

```
% Example 8.4 Apply the Blackman window to the rectangular window impulse
responses
%  developed in Example 8.3.
%
% Generate the filter coefficients
..... same code as in Example 8.3.......
     if n == 0          % Generate sin(n)/n function.  Use Equation 8.28.
        b(k) = 2*fc;    % Case where denominator is zero.
     else
        b(k) = sin(2*pi*fc*n)/(pi*n);   % Filter impulse response
     end
  w = blackman(L);      % Get Blackman window of length L
  b =  b .* w ;         % Apply window to impulse response
  H = fft(b,N);         % Calculate spectrum
......same code as in Example 8.3, plot and label.......
```

[8]This equation is written assuming the popular value for constant α of 0.16.

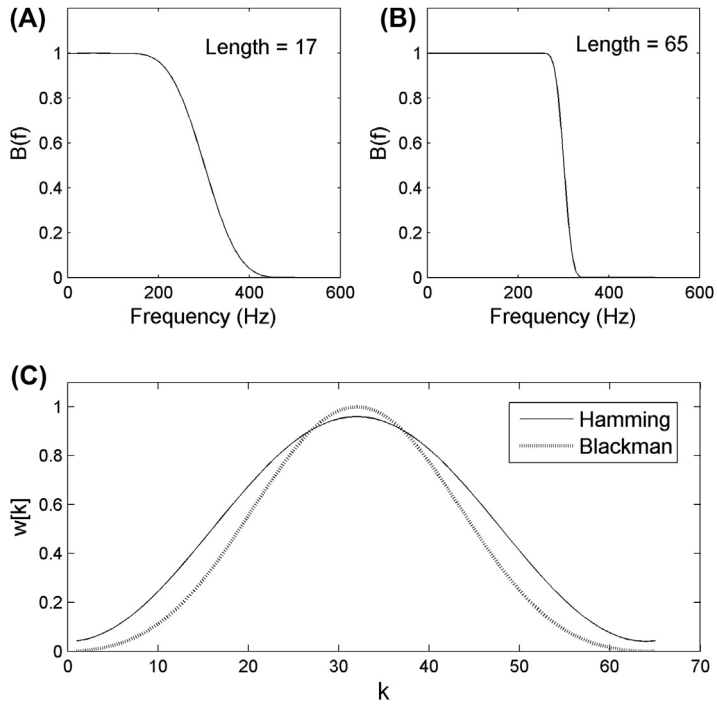

FIGURE 8.14 (A) Magnitude spectrum produced by the 17-coefficient finite impulse response (FIR) filter in Figure 8.13 except a Blackman window was applied to the filter coefficients. The Gibbs oscillations seen in Figure 8.13 are no longer visible. (B) Magnitude spectrum produced by the 65-coefficient FIR filter in Figure 8.13 also after application of the Blackman. (C) Plot of the Blackman and Hamming window functions. Both have cosine-like appearances.

```
end

function w = blackman(L)
% Function to calculate a Blackman window L samples long
%
n = (1:L);          % Generate vector for window function
w = 0.42 - 0.5*cos(2*pi*n/(L-1)) + 0.08*cos(4*pi*n/(L-1));
```

Results: The Blackman window is easy to generate in MATLAB, and when applied to the impulse responses of Example 8.3, substantially reduces the oscillations as shown in Figure 8.14. The filter rolloff is still not that of an ideal filter, but becomes steeper for a longer filter length. Of course, increasing the length of the filter increases the computation time required to apply the filter to a data set, a typical engineering compromise. Figure 8.14C shows a plot of the Blackman and Hamming windows. Both of these popular windows are quite similar since they use raised cosines to taper the filter coefficients.

The next example applies a rectangular window low-pass filter to a signal of human respiration that was obtained from a respiratory monitor.

EXAMPLE 8.5

Apply a low-pass rectangular window filter to the 10-min respiration signal shown in the top trace of Figure 8.15. This signal is found as variable `resp` in file `Resp.mat`. $f_s = 12.5$ Hz. The sampling frequency is low because the respiratory signal has a very low bandwidth. Use a cutoff frequency of 1.0 Hz and a filter length of 65. Use the Blackman window to truncate the filter's impulse response. Plot the original and filtered signal.

Solution: Reuse the code in Examples 8.3 and 8.4 and apply the filter to the respiratory signal using convolution. MATLAB's `conv` routine is invoked with noncausal filtering by using the option `'same'` to avoid a time shift in the output.

```
% Example 8.5  Application of a rectangular window low-pass filter to a respiratory
signal.
%
load Resp;                    % Get data
fs = 12.5;                    % Sampling frequency
N = length(resp);             % Get data length
t = (1:N)/fs;                 % Time vector for plotting
L = 65;                       % Filter lengths
fc = 1/fs;                    % Cutoff frequency: 1.0 Hz
plot(t,resp+1,'k'); hold on;        % Plot original data (offset for clarity)
for k = 1:L                         % Generate sin(n)/n function symmetrically
  n = k-(L-1)/2 ;                   % Same code as in previous examples
  if n == 0
    b(k) = 2*fc;                    % Case where denominator is zero.
  else
    b(k) = (sin(2*pi*fc*n))/(pi*n); % Filter impulse response
  end
end
b = b.*blackman(L);                 % Apply Blackman window
%
y = conv(resp,b,'same');            % Apply filter
plot(t,y,'k');
```

Results: The filtered respiratory signal is shown in the lower trace of Figure 8.15. The filtered signal is smoother than the original signal as the noise riding on the original signal has been eliminated.

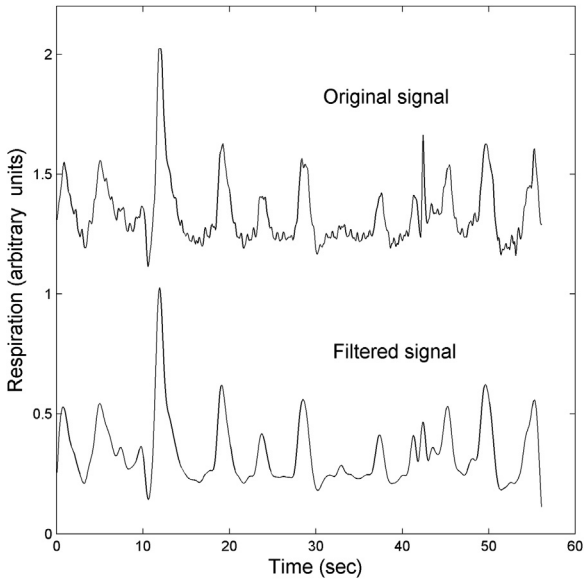

FIGURE 8.15 The output of a respiratory monitor is shown in the upper trace to have some higher frequency noise. In the lower trace, the signal has been filtered with a finite impulse response rectangular low-pass filter. The filter has a cutoff frequency of 1.0 Hz and a filter length of 65. *Original data from PhysioNet, Goldberger, A.L., Amaral, L.A.N., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.-K., Stanley, H.E., 2000. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. Circulation 101 (23), e215—e220. Circulation Electronic Pages. http://circ.ahajournals.org/cgi/content/full/101/23/e215.*

The FIR filter coefficients for high-pass, band-pass, and band-stop filters can also be derived by applying an inverse Fourier transform to rectangular spectra having the appropriate associated shape. These equations have the same general form as Equation 8.28 except they may include additional terms:

$$b[k] = \begin{cases} -\dfrac{\sin[2\pi f_c(k-L/2)]}{\pi(k-L/2)} & k \neq \dfrac{L}{2} \\ 1-2f_c & k = \dfrac{L}{2} \end{cases} \quad \text{Highpass} \tag{8.33}$$

$$b[k] = \begin{cases} \dfrac{\sin[2\pi f_h(k-L/2)]}{\pi(k-L/2)} - \dfrac{\sin[2\pi f_l(k-L/2)]}{\pi(k-L/2)} & k \neq \dfrac{L}{2} \\ 2(f_h - f_l) & k = \dfrac{L}{2} \end{cases} \quad \text{Bandpass} \tag{8.34}$$

$$b[k] = \begin{cases} \dfrac{\sin[2\pi f_l(k-L/2)]}{\pi(k-L/2)} - \dfrac{\sin[2\pi f_h(k-L/2)]}{\pi(k-L/2)} & k \neq \dfrac{L}{2} \\ 1 - 2(f_h - f_l) & k = \dfrac{L}{2} \end{cases} \quad \text{Bandstop} \tag{8.35}$$

The order of high-pass and band-stop filters should always be even, so the number of coefficients in these filters should be odd. The next example applies a band-pass filter to the electroencephalogram (EEG) signal introduced in Chapter 1.

EXAMPLE 8.6

Apply a band-pass filter to the `EEG` data in file `ECG.mat`. Use a lower cutoff frequency of 6 Hz and an upper cutoff frequency of 12 Hz. Use a Blackman window to truncate the filter's impulse to 129 coefficients. Plot the data before and after band-pass filtering. Also plot the spectrum of the original signal and superimpose the spectrum of the band-pass filter.

Solution: Construct the filter's impulse response using the band-pass equation shown in Equation 8.34. Note the special case where the denominator in the equation goes to zero (i.e., when $k = L/2$). The application of limits and the small angle approximation ($\sin(x) \rightarrow x$) gives a coefficient value of $b[L/2] = 2f_h - 2f_l$, as shown in Equation 8.33. Recall that the sampling frequency of the EEG signal is 100 Hz.

After applying the filter and plotting the resulting signal, compute the signal spectrum using the Fourier transform and plot. The filter's spectrum is found by taking the Fourier transform of the impulse response ($b[k]$) and is plotted superimposed on the signal spectrum.

```
% Example 8.6 Apply a band-pass filter to the EEG data in file ECG.mat.
%
load EEG;                    % Get data
N = length(eeg);             % Get data length
fs = 100;                    % Sample frequency
fh = 12/fs;                  % Set highpass and
fl = 6/fs;                   % lowpass cutoff frequencies
L = 129;                     % Set number of weights
%
for k = 1:L                  % Generate bandpass filter coefficients

  n = k - (L-1)/2 ;          % Make symmetrical
  if n == 0
    b(k) = 2*fh - 2*fl;      % Case where denominator is zero
  else
    b(k) = sin(2*pi*fh*n)/(pi*n) - sin(2*pi*fl*n)/(pi*n) ;  % Filter coefficients
  end
end
b = b .* blackman(L);        % Apply Blackman window
y = conv(eeg,b,'same');      % Filter the data using convolution
  .......plot eeg before and after filtering; plot eeg and filter spectrum......
```

Results: Band-pass filtering the EEG signal between 6 and 12 Hz reveals a strong higher-frequency oscillatory signal that is washed out by lower frequency components in the original

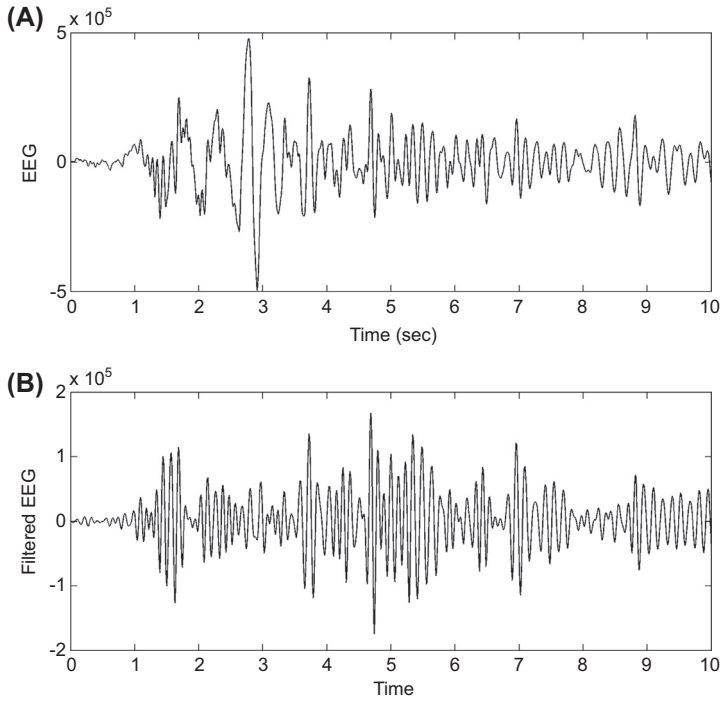

FIGURE 8.16 Electroencephalogram signal before (A) and after (B) band-pass filtering between 6 and 12 Hz. A fairly regular oscillation is seen in the filtered signal.

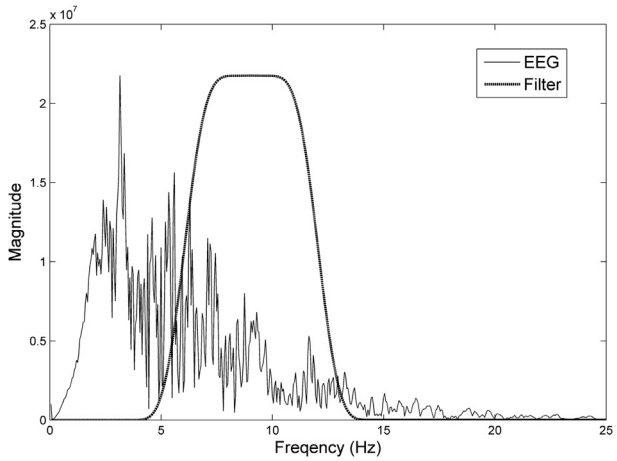

FIGURE 8.17 The magnitude spectrum of the electroencephalography signal used in Example 8.5 along with the spectrum of a band-pass filter based on Equation 8.12. The band-pass filter range is designed to be between 6 and 12 Hz.

signal as shown in Figure 8.16B. This figure shows that filtering can significantly alter the appearance and interpretation of biomedical data. The band-pass spectrum shown in Figure 8.17 has the desired cutoff frequencies and, when compared with the EEG spectrum, is shown to reduce the high- and low-frequency components of the EEG signal.

Implementation of other FIR filter types is found in the problem set. A variety of FIR filters exist that use strategies other than the rectangular window to construct the filter coefficients, and some of these are explored in the section on MATLAB implementation. One FIR filter of particular interest is used to construct the derivative of a waveform, since the derivative is often of interest in the analysis of biosignals. The next section explores a popular filter for this operation.

8.5.1 Derivative Filters—The Two-Point Central Difference Algorithm

The derivative is a common operation in signal processing, and is particularly useful in analyzing certain physiological signals. Digital differentiation is defined as $dx[n]/dn$ and can be calculated directly from the slope of $x[n]$ by taking differences:

$$\frac{dx[n]}{dn} = \frac{\Delta x[n]}{T_s} = \frac{x[n+1] - x[n]}{T_s} \tag{8.36}$$

This equation can be implemented by MATLAB's `diff` routine. This routine uses no padding, so the output is one sample shorter than the input. As shown in Chapter 6 (Section 6.5.2), the frequency characteristic of the derivative operation increases linearly with frequency, so differentiation enhances higher frequency signal components (see Figure 6.11). Since the higher frequencies frequently contain a greater percentage of noise, this operation tends to produce a noisy derivative curve. The upper curve of Figure 8.18A is a fairly clean physiological motor response, an eye movement response similar to that used in Figure 8.10. The lower curve of Figure 8.18A is the velocity of the movement obtained by calculating the derivative using MATLAB's `diff` routine which implements Equation 8.36. Considering the relative smoothness of the original signal, the velocity curve obtained using Equation 8.36 is quite noisy.

In the context of FIR filters, Equation 8.36 is equivalent to a two-coefficient filter: $b[k] = [+1/T_s, -1/T_s]$. (Note that the positive and negative coefficients are reversed by convolution, so they are sequenced in reverse order in the impulse response.) A better approach to differentiation is to construct a filter that approximates the derivative at lower frequencies but attenuates higher frequencies that are likely to be only noise. The "two-point central difference algorithm" achieves such an effect, acting as a differentiator at lower frequencies and a low-pass filter at higher frequencies. Figure 8.18B shows the same responses when this algorithm is used to estimate the derivative. The result is a much cleaner velocity signal that still captures the peak velocity of the response.

The two-point central difference algorithm still subtracts two points to get a slope but the two points are no longer adjacent, rather, they may be spaced some distance apart. Putting this in FIR filter terms, the algorithm is based on an impulse function containing two coefficients of equal but opposite sign, spaced L points apart. The equation defining the filter coefficients of this differentiator is:

$$\frac{dx[n]}{dn} = \frac{x(n+L) - x(n-L)}{2LT_s} \tag{8.37}$$

where L is now called the "skip factor" that defines the distance between the points used to calculate the slope, and T_s is the sample interval. The skip factor, L, influences the effective

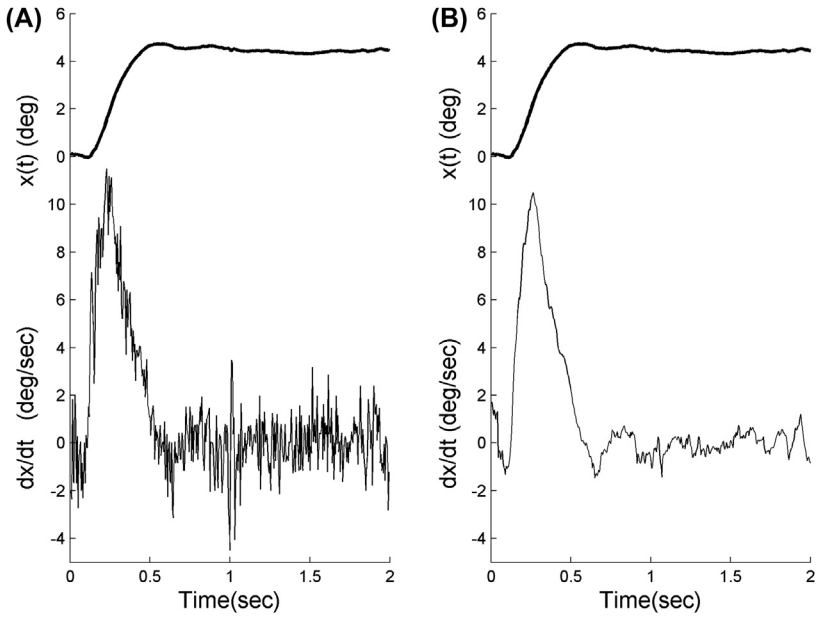

FIGURE 8.18 An eye movement response to a step change in target depth is shown in the upper trace, and its velocity (i.e., derivative) is shown in the lower trace. (A) The derivative was calculated by taking the difference in adjacent points and scaling by the sample interval, following Equation 8.36. The velocity signal is noisy, even though the original signal is fairly smooth. (B) The derivative was computed using the two-point central difference algorithm with a skip factor of four. This filter results in a much cleaner derivative signal.

bandwidth of the filter as shown below. Implemented as an FIR filter, Equation 8.37 leads to coefficients:

$$b[k] = \begin{cases} 1/2LT_s & k = -L \\ -1/2LT_s & k = +L \\ 0 & k \neq \pm L \end{cases} \tag{8.38}$$

Again, note that the $+L$ coefficient is negative and the $-L$ coefficient is positive since the convolution operation reverses the order of $b[k]$. As with all FIR filters, the frequency response of this filter algorithm can be determined by taking the Fourier transform of $b[k]$. Since this function is fairly simple, it is not difficult to take the Fourier transform analytically (trying to break from MATLAB wherever possible) as well as in the usual manner using MATLAB. Both methods are presented in the following example.

EXAMPLE 8.7

(A) Determine the magnitude spectrum of the two-point central difference algorithm analytically, then (B) use MATLAB to determine the spectrum and apply it to the EEG signal.

Analytical Solution: Starting with the equation for the discrete Fourier transform (Equation 3.34) substituting k for n:

$$X[m] = \sum_{k=0}^{N-1} b[k]\, e^{-j2\pi mk/N}$$

Since $b[k]$ is nonzero only for $k = \pm L$, the Fourier transform, after the summation limits are adjusted for a symmetrical coefficient function with positive and negative n, becomes:

$$X(m) = \sum_{k=-L}^{L} b[k] e^{-j2\pi mk/N} = \frac{1}{2LT_s} e^{-j2\pi m(-L)/N} - \frac{1}{2LT_s} e^{-j2\pi mL/N}$$

$$X(m) = \frac{e^{-j2\pi m(-L)/N} - e^{-j2\pi mL/N}}{2LT_s} = \frac{-j\sin(2\pi mL/N)}{LT_s}$$

where L is the skip factor and N is the number of samples in the waveform. To put this equation in terms of frequency, note that $f = m/(N\,T_s)$, hence $m = f\,N\,T_s$.

To find $|X(f)|$, substitute $f\,N\,T_s$ for m and take the magnitude.

$$|X(f)| = \left| -j\frac{\sin(2\pi fLT_s)}{L\,T_s} \right| = \frac{|\sin(2\pi fLT_s)|}{L\,T_s}$$

This equation shows that the magnitude spectrum, $|X(f)|$, is a sine function that goes to zero at $f = 1/(LT_s) = f_s/L$. Figure 8.19 shows the frequency characteristics of the two-point central difference algorithm for two different skip factors: $L = 2$ and $L = 6$.

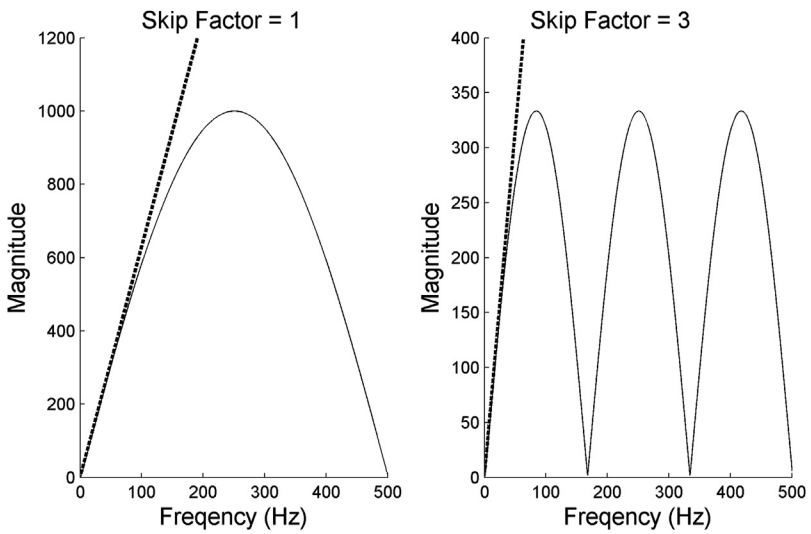

FIGURE 8.19 The frequency response of the two-point central difference algorithm using two different skip factors: (A) $L = 2$; (B) $L = 6$. The *dashed line* shows the frequency characteristic of a simple differencing operation. The sample frequency is 1.0 kHz.

MATLAB Solution: Finding the spectrum using MATLAB is straightforward once the coefficient vector is constructed. An easy way to construct the impulse response is to use brackets and concatenate zeros between the two end values, essentially following Equation 8.38 directly. Again, the initial filter coefficient is positive and the final coefficient negative to account for the reversal produced by convolution.

```
% Example 8.7  Determine the frequency response of
%  the two point central difference algorithm used for differentiation.
%
Ts = .001;                  % Assume a Ts of 1 msec. (i.e., fs = 1 kHz)
N = 1000;                   % Number of data points (time = I sec)
Ln = [1 3];                 % Define two different skip factors
for skip = 1:2              % Repeat for each skip factor
  L = Ln(skip);             % Set skip factor
  b = [1/(2*L*Ts) zeros(1,2*L-1) -1/(2*L*Ts)];   % Filter impulse response
  H = abs(fft(b,N));        % Calculate magnitude spectrum
  ….. plot and label spectrum; plot straight line for comparison….
end
```

Result: The result of both this program and the analysis are shown in Figure 8.19. A true derivative has a linear change with frequency: a line with a slope proportional to f as shown by the dashed lines in Figure 8.19. The two-point central difference spectrum approximates a true derivative over the lower frequencies, but has the characteristic of a low-pass filter for higher frequencies. Increasing the skip factor, L, has the effect of lowering the frequency range over which the filter acts like a derivative operator as well as lowering the low-pass filter range. Note that for skip factors >2, the response curve repeats above $f = 1/(LT_s)$. Usually the assumption is made that the signal does not contain frequencies in this range. If this is not true, then these higher frequencies can be removed by an additional low-pass filter as shown in one of the problems. It is also possible to combine the difference equation (Equation 8.15) with a low-pass filter and this is also explored in one of the problems.

8.5.2 Determining Cutoff Frequency and Skip Factor

Determining the appropriate cutoff frequency of a filter or the skip factor for the two-point central difference algorithm can be somewhat of an art (meaning there is no definitive approach to a solution). If the frequency ranges of the signal and noise are known, setting cutoff frequencies is straightforward. But this knowledge is not usually available in biomedical engineering applications. In most cases, filter parameters such as filter order and cutoff frequencies are set empirically based on the data. In one trial-and-error scenario, the signal bandwidth is progressively reduced until some desirable feature of the signal is lost or compromised. Although it is not possible to establish definitive rules because of the task-dependent nature of filtering, the next example gives an idea about how these decisions are approached.

When taking derivatives, we often strive to remove the most noise while still preserving the derivative peaks. For example, if the signal represents a movement, then its derivative

is velocity, and we frequently want a good measure of peak velocity. In the next example, we return to the eye movement signal shown in Figure 8.18 to evaluate several different skip factors to find the one that gives the best reduction of noise without reducing the accuracy of the velocity trace. We use a strictly empirical approach: we increase the skip factor until the measurement of peak velocity is reduced.

EXAMPLE 8.8

Use the two-point central difference algorithm to compute velocity traces of the eye movement step response in file `eye.mat`. Use four different skip factors (1, 2, 5, and 10) to find the skip factor that best reduces noise without substantially reducing the peak velocity of the movement.

Solution: Load the file and use a loop to calculate and plot the velocity determined with the two-point central difference algorithm using the four different skip factors. Find the maximum value of the velocity trace for each derivative evaluation and display on the associated plot. The original eye movement shown in Figure 8.18 (upper traces), is in degrees so the velocity trace is in degrees per second.

```
% Example 8.8  To evaluate the two-point central difference algorithm using
%  different derivative skip factors
%
load eye;                                % Response in vector eye_move
fs = 200;                                % Sampling frequency
Ts = 1/fs;                               % Calculate Ts
t = (1:length(eye_move))/fs;             % Time vector for plotting
L = [1 2 5 10];                          % Filter skip factors
for skip = 1:4                           % Loop for different skip factors
  b  =  [1/(2*L(skip)*Ts)  zeros(1,2*L(skip)-1)  -1/(2*L(skip)*Ts)];  %  Construct
filter
  der = conv(eye_move,b,'same');         % Apply filter
  subplot(2,2,skip);
  plot(t,der,'k');                       % Plot velocity curve
.....text, labels, and axis.......
end
```

Results: The results of this program are shown in Figure 8.20. As the skip factor increases, the noise decreases, and the velocity trace becomes quite smooth at a skip factor of 10. But our measurement of peak velocity also decreases at the higher skip factors. Deciding on the best skip factor requires that we exercise some judgment. Examining the curve with the lowest skip factor ($L = 1$) suggests that the peak velocity shown, 23 degrees/s, may be augmented a bit by noise. A peak velocity of 21 degrees/s is found for both skip factors of 2 and 5. This peak velocity appears to be reflective of the true peak velocity. The peak found using a skip factor of 10 is clearly reduced. A skip factor of around 5 seems appropriate, but this is a judgment call. In fact, judgment and intuition are all too frequently involved in signal processing and signal analysis tasks, a

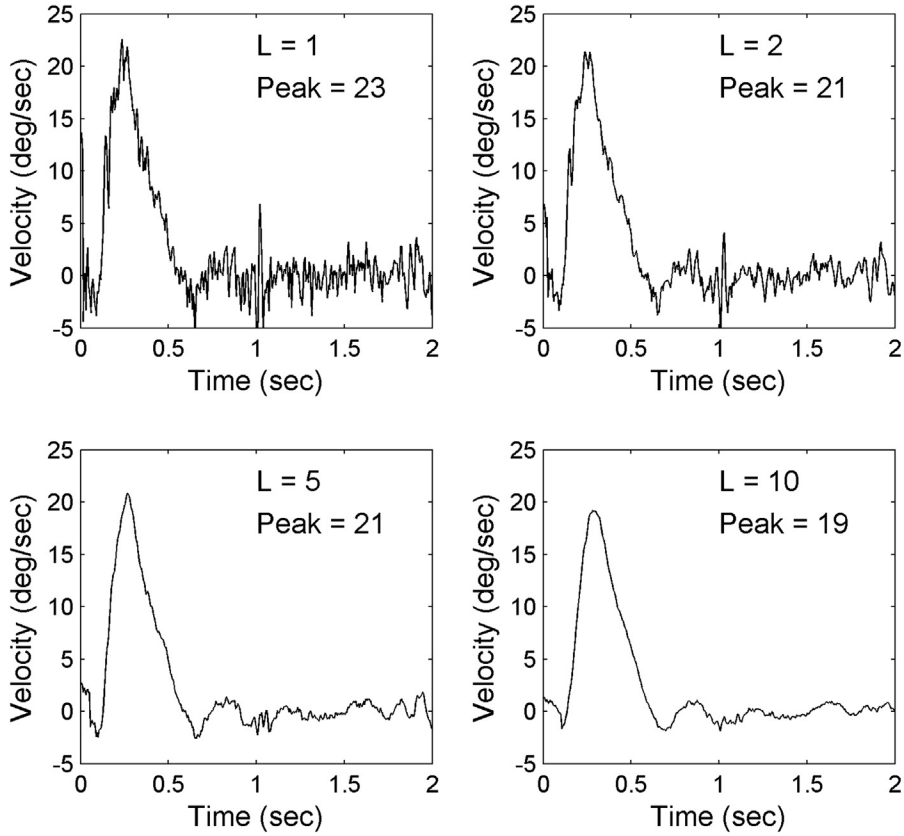

FIGURE 8.20 Velocity traces for the eye movement shown in Figure 8.18, calculated by the two-point central difference algorithm for different values of skip factor as shown. The peak velocities are shown in degrees per second.

reflection that signal processing is still an art. Other examples that require empirical evaluation are given in the problem set.

8.6 FINITE IMPULSE RESPONSE AND INFINITE IMPULSE RESPONSE FILTER DESIGN USING THE SIGNAL PROCESSING TOOLBOX

Unlike FIR filters, IIR filters are quite difficult to design from first principles. MATLAB again comes to our rescue with the Signal Processing Toolbox. This toolbox offers considerable support for the design and evaluation of both FIR and IIR filters. The remainder of this chapter is devoted to using this toolbox to design both simple and more complicated filters.

Within the MATLAB environment, filter design and application occur in either one or two stages, each stage executed by separate but related routines. In the two-stage protocol, the user supplies information regarding the filter type and desired attenuation characteristics,

but not the filter order. The first-stage routines determine the appropriate order as well as other parameters required by the second-stage routines. The second-stage routines then generate the filter coefficients, $b[k]$, based on the arguments produced by the first-stage routines, including the filter order.

It is possible to bypass the first stage routines if you already know, or can guess, the filter order. Only the second-stage routines are needed. In this situation the user supplies the filter order along with other filter specifications. In practical situations, trial and error is often required to determine the filter order and first stage routines are not helpful. For this reason, we ignore first-stage routines, and discuss only the second-stage filter design routines.

Other tools that we will not describe include an interactive filter design package called `FDATool` (for Filter Design and Analysis Tool) that uses a graphical user interface to design filters with highly specific or demanding spectral characteristics. Yet another Signal Processing Toolbox package, the `SPTool` (Signal Processing Tool), is useful for analyzing filters and generating spectra of both signals and filters. The following basic routines should cover all your filtering needs, but you can always go to MATLAB `help` for detailed information on these and other packages.

Irrespective of how we get there, the net result is a set of b and a coefficients that uniquely define the filter's spectrum. From there, we already know how to apply the filter to a signal using either the `conv` or `filter` routine. Alternatively the Signal Processing Toolbox contains a third routine, `filtfilt` that, like `conv` with the `'same'` option, employs noncausal methods to implement filters that have no time delay. However, `filtfilt` can also be used to implement IIR filters. The calling structure is exactly the same as `filter`:

```
y = filtfilt(b,a,x);      % Noncausal IIR filter applied to signal x
```

One of the problems dramatically illustrates the difference between the use of `filter` and `filtfilt`.

As a brief aside, there is a useful Signal Processing Toolbox routine that determines the frequency response of a filter given the coefficients. We already know how to do this, we did it in Example 8.1 using the Fourier transform (i.e., Equation 8.15). But the MATLAB routine `freqz` also includes frequency scaling and plotting, making it quite convenient.

```
[H,f] = freqz (b,a,n,fs);
```

where `b` and `a` are the filter coefficients and `n` is optional and specifies the number of points in the desired frequency spectra. For FIR filters, the value of `a` is set to 1.0 as in the `filter` routine. The input argument, `fs`, is also optional and specifies the sampling frequency. Both output arguments are also optional and are usually not given. If `freqz` is called without the output arguments, the magnitude and phase plots are produced automatically. If the output arguments are specified, the output vector `H` is the complex frequency response of the filter (the same variable produced by `fft`). The second optional output `f` is a frequency vector useful in plotting. If `fs` is given, `f` is in hertz and ranges between 0 and $f_s/2$; otherwise a less useful `f` is provided in radians per sample and ranges between 0 and π.

8.6.1 Finite Impulse Response Filter design

Although we already know how to design the basic rectangular window FIR filters, the Signal Processing Toolbox makes it so easy that it is hard to resist. The toolbox supports all rectangular window FIR filters: low pass, high pass, band pass, and band stop (Equations 8.28, 8.33–8.35). The calling structure is:

```
b = fir1(N,Wn,'type');    % Design a rectangular window filter
```

where `b` is a vector containing the coefficients, `N` is the filter order, and `Wn`[9] is the cutoff frequency (with respect to $f_s/2$, not f_s, so when `wn` = 1, $f_c = f_s/2$). For a low-pass filter, the optional variable `'type'` is absent. If you want a band-pass filter, simply make `Wn` a two-element vector that contains low and high cutoff frequencies, again relative to $f_s/2$. If you want a high-pass filter, make the third variable `'high'` and use with a single `Wn` to specify the cutoff frequency. Finally for a band-stop filter, again make Wn a two-element variable specifying the lower and upper cutoff frequencies and make the third variable 'stop.' An optional fourth variable can be used to specify a tapering window (which must be `N` + 1 points long), but if it is omitted, the filter coefficients are tapered by the ever popular Hamming window. Use `help fir1` to find other features you are unlikely to need.

In some rare cases, there may be a need for a filter with a more exotic spectrum for which we turn to `fir2`. This routine produces an FIR filter with a spectrum of almost any shape. The command structure for `fir2` is:

```
b = fir2(N,F,A)
```

where `N` is the filter order, `F` is a vector of normalized frequencies in ascending order, and `A` is the desired gain of the filter at the corresponding frequency in vector `F`. In other words, `plot(F,A)` would show the desired magnitude frequency curve. Clearly `F`and `A` must be the same length, but duplicate frequency points are allowed corresponding to step changes in the frequency response. In addition, the first value of `f` must be 0.0 and the last value 1.0 (equal to $f_s/2$). Some additional optional input arguments are mentioned in the MATLAB `help` file. The next example shows the use of `fir1` and the flexibility of `fir2`.

EXAMPLE 8.9

Use `fir1` to design a band-pass filter with a passband between 50 and 100 Hz. Use `fir2` to design a second filter, a double band-pass filter with one passband also between 50 and 100 Hz and a second passband between 150 and 200 Hz. Make the order 65 for both filters. Apply this filter to a signal containing sinusoids at 75, and 175 Hz buried in 20 dB of noise (SNR = −20 dB). Use routine `sig_noise` (Section 3.4.3) to generate a 2000-sample signal. (Recall `sig_noise` assumes $f_s = 1$ kHz.)

[9]Normally, we use f for frequency in hertz and ω for frequency in radians. But here `Wn` is frequency in hertz, although it is a relative frequency (relative to $f_s/2$). It is confusing to use Wn for frequency in hertz, but MATLAB uses it in most of their filter routines, so we stick with it to be consistent with MATLAB. Yet in routine `fir2`, described next, MATLAB uses F for relative frequency in hertz. So much for consistency.

Plot the magnitude spectrum of both filters and the magnitude spectrum of the signal before and after filtering with both filters.

Solution: Constructing a single band-pass filter using `fir1` is straightforward. We could construct a double band-pass filter by putting two such band-pass filters in series, but instead we can use `fir2` to construct a single FIR filter having the desired filter characteristics.

For the double band-pass filter, we first specify the desired frequency characteristics in terms of a frequency and a gain vector. The frequency vector must begin with 0.0 and end with 1.0, but can have duplicate frequency entries to allow for step changes in gain. We can apply these filters in a causal manner using either `filter` or `conv`. In this example, we use `filter` for a little variety. Then, the magnitude spectra of the filtered and unfiltered waveforms are determined using the Fourier transform.

```
% Example 8.9 Design a double bandpass filter.
fs = 1000;                % Sample frequency
N = 2000;                 % Number of points
Nf = 65;                  % Filter order
% Cutoff frequencies of single bandpass filter
fl1 = 50/(fs/2);          % First peak low cutoff freq.
fh1 = 100/(fs/2);         % First peak high cutoff freq.
fl2 = 150/(fs/2);         % Second peak low freq. cutoff
fh2 = 200/(fs/2);         % Second peak high freq. cutoff
%
x = sig_noise([75 175],-20,N);              % Generate noise waveform
%
b1 = fir1(Nf,[fl1,fh1]);                    % Design single bandpass filter.
% Design double bandpass filter
F = [0 fl1 fl1 fh1 fh1 fl2 fl2 fh2 fh2 1]; % Construct desired
A = [0  0  1  1  0  0  1  1  0  0];         %  frequency/gain characteristic
b2 = fir2(Nf,F,A);
[H1,f1] = freqz(b1,1,512,fs);               % Calculate filter1 frequency response
H2 = freqz(b2,1,512,fs);                    % Calculate filter2 frequency response
%
y1 = filter(b1,1,x);                        % Apply single bandpass filter
y2 = filter(b2,1,x);                        % Apply double bandpass filter
Xf = abs(fft(x));                           % Get signal spectrum
Yf1 = abs(fft(y1));                         % Get spectrum of filtered signals
Yf2 = abs(fft(y2));
.......plot and label magnitude spectra.......
```

Results: Figure 8.21 is a plot of the magnitude spectrum of the noisy signal. It looks noisy with no hint of the buried sinusoids. Figure 8.22 shows the magnitude spectra (upper curves) of the single and double band-pass filters. The lower curves of Figure 8.22 show the spectra of the noisy signal after filtering by the two filters, and the signal peaks are clearly visible. These peaks are not any larger than they were in Figure 8.21, but are much more evident because the surrounding energy, energy outside the various passbands, has been attenuated.

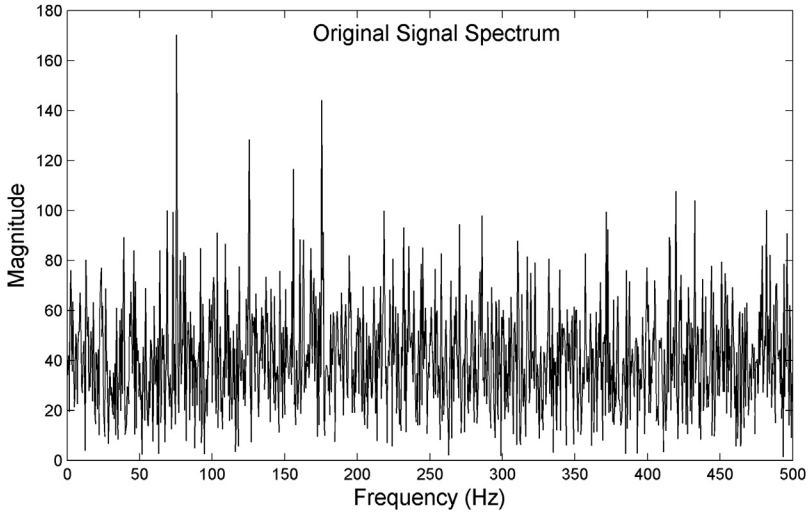

FIGURE 8.21 The magnitude spectra of the noisy signal used in Example 8.9. There are two sinusoids in this signal buried in 20 dB of noise; that is, the signal to noise ratio is −20 dB.

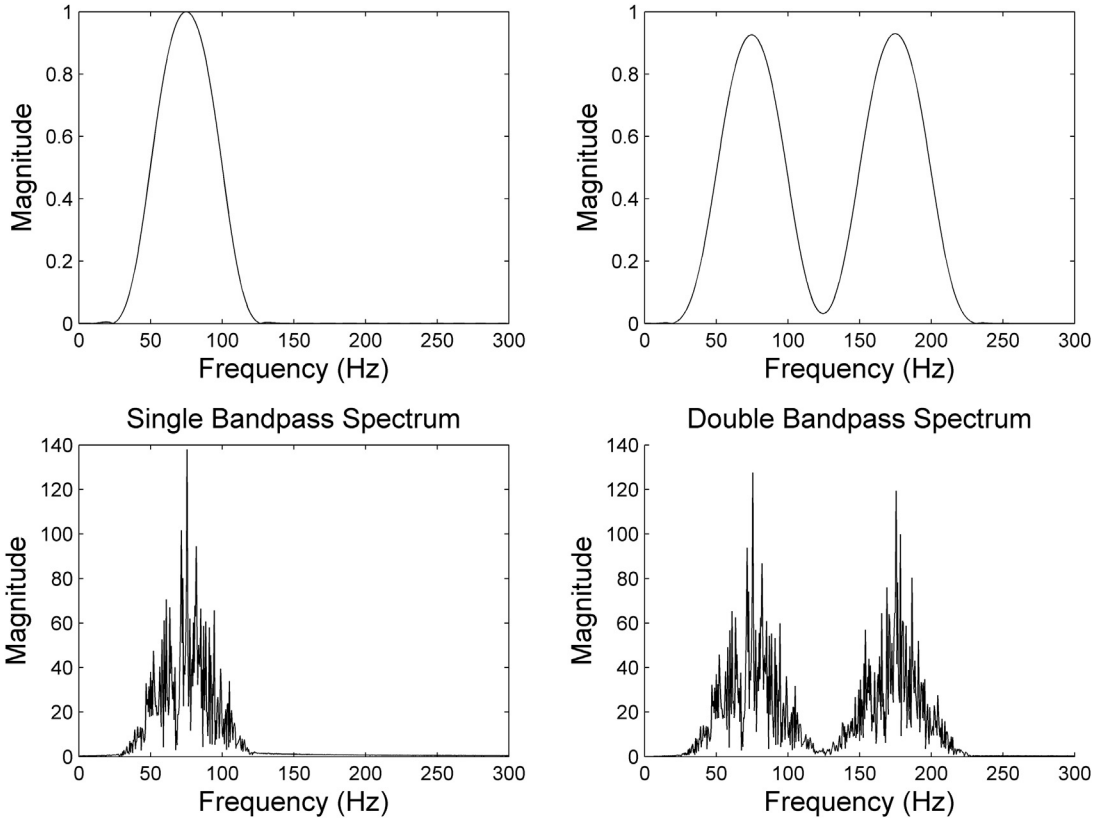

FIGURE 8.22 The spectra (upper curves) of the single and double band-pass filters used in Example 8.9. The lower curves show the spectra of the noisy signal after filtering. Although the signal peaks are not really any larger than they are in Figure 8.21, they are much more apparent because of the attenuation of the surrounding noise.

Although the effect of filtering in this example is quite dramatic, we set the filter's frequency ranges to center on the signal frequencies. This can be done only if we know the frequencies we are trying to isolate ahead of time.

8.6.2 Designing Infinite Impulse Response Filters

IIR filters can achieve sharp rolloffs with far fewer filter coefficients than FIR filters (e.g., see Figure 8.8). For standard noise filtering of one-dimensional signals, the IIR filter is usually your go-to filter. The design of IIR filters is not as straightforward as that for FIR filters and whole books have been written on this topic. But with the MATLAB Signal Processing Toolbox, anyone can do it. Although you need the Signal Processing Toolbox to design IIR filters, you do not need it to apply these filters. IIR filters are totally specified by their a and b coefficients, so once you have these coefficients, you can apply the filters using the standard `filter` routine, or with any other implementation of Equation 8.21. You could use another programming language to implement these filters; you do not need MATLAB.

IIR filters are analogous to some popular analog filters and originally were designed using tools developed for analog filters. In an analog filter, there is a direct relationship between the number of independent energy storage elements in the system and the filter's rolloff slope: each energy storage element increases the downward (or upward) slope by 20 dB/decade. In IIR filters, the first a coefficient, $a[0]$, always equals 1.0, but each additional a coefficient adds 20 dB/decade to the attenuation slope. So, an eighth-order IIR filter with nine a coefficients has the same attenuation characteristics as an eighth-order analog filter.[10] Since the slope of an IIR filter increases by 20 dB/decade for each filter order, determining the filter order needed for a given desired attenuation is straightforward.

IIR filter design under MATLAB follows the same procedures as FIR filter design, only the names of the routines are different. In the MATLAB Signal Processing Toolbox, the two-stage design process is supported for most of the IIR filter types. But as with FIR design, a single-stage design process can be used provided you specify the filter order. Again, we stick to single-stage design as is commonly used practice.

The Yule-Walker recursive filter is the IIR equivalent of the `fir2` FIR filter routine in that it allows for the specification of a flexible frequency characteristic. The calling structure is also very similar to that of `fir2`.

```
[b,a] = yulewalk(N,F,A);
```

where `N` is the filter order, and `F` and `A` specify the desired frequency characteristic in the same manner as `fir2`: `A` is a vector of the desired filter gains at the frequencies specified in `F`. The `F` and `A` vectors follow the same rules as in `fir2`: frequencies are relative to $f_s/2$, the

[10]In the analog world, one rarely finds a filter higher than eighth-order. Even using integrated circuits, the circuitry becomes overly complex. Moreover, an eighth-order filter is usually sufficient, even for demanding filtering tasks, such as antialiasing. In fact, lower order filters such as fourth-order analog filters are often used in many analog filtering applications.

first point in `F` must be 0, the last point 1.0, and duplicate frequency points are allowed. Duplicate frequency points enable step changes in the frequency response. This routine is used in the next example.

EXAMPLE 8.10

Design the double band-pass filter that is used in Example 8.9 with one passband, also between 50 and 100 Hz and a second passband between 150 and 200 Hz. Use a 12th-order IIR filter and compare the results with the 65th-order FIR filter used in Example 8.9. Plot the frequency spectra of both filters superimposed for easy comparison.

Solution: Modify Example 8.9 to add the Yule-Walker filter, determine its spectrum using `freqz`, and plot superimposed on the spectrum of the double band-pass FIR filter. Remove the code relating to the single band-pass filter and the signal filtering as it is not needed in this example.

```
% Example 8.10 Design a double bandpass IIR filter and compare with a similar
FIR filter
%
  ..... same initial code as in Example 8.9.....
N1 = 65;                % FIR Filter order
N2 = 12;                % IIR Filter order
fl1 = 50/(fs/2);        % First peak low cutoff freq.
fh1 = 100/(fs/2);       % First peak high cutoff freq.
fl2 = 150/(fs/2);       % Second peak low freq. cutoff
fh2 = 200/(fs/2);       % Second peak high freq. cutoff
%
% Design filter
F = [0 fl1 fl1 fh1 fh1 fl2
fl2 fh2 fh2 1];                     % Construct desired
A = [0  0  1  1  0  0  1  1  0  0];     %  frequency characteristic
b1 = fir2(N1,F,A);                  % Construct FIR filter
[b2 a2] = yulewalk(N2,F,A);         % Construct IIR filter
[H1,f1] = freqz(b1,1,512,fs);       % Calculate FIR frequency response
[H2 f2] = freqz(b2,a2,512,fs);      % Calculate IIR frequency response
       .......plots and labels.......
```

Results: The spectral results from two filters are shown in Figure 8.23. The magnitude spectra of both filters look quite similar despite the fact that the IIR filter has far fewer coefficients. (The FIR filter has 66 b coefficients, whereas the IIR filter has 13 b coefficients and 13 a coefficients.)

In an extension of Example 8.10, the FIR and IIR filters are applied to a random signal of 10,000 data points, and MATLAB's `tic` and `toc` are used to evaluate the time required for each filter operation.

```
% Example 8.10 (continued) Time required to apply the two filters in Example 8.10
%  to a random array of 10,000 samples.
```

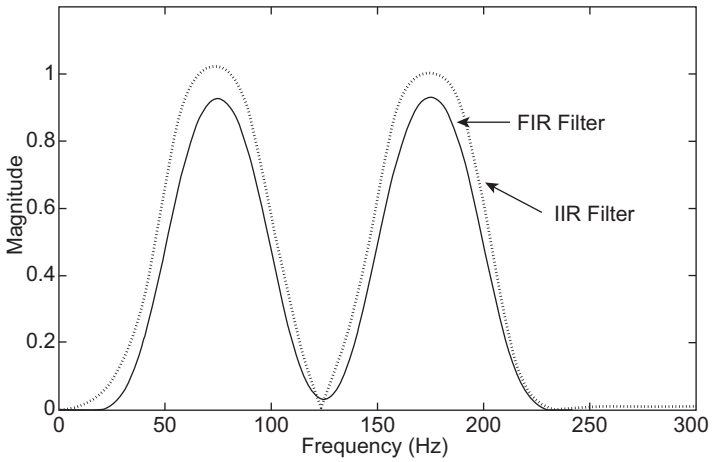

FIGURE 8.23 The filter spectra of two double band-pass filters, a 65th-order finite impulse response (FIR) used in Example 8.9, and a 12th-order infinite impulse response (IIR) filter. The two spectra are quite similar despite the large difference in the number of filter coefficients. (The FIR filter has 66 coefficients, whereas the IIR filter has 13 coefficients.)

```
%
x = rand(1,10000);          % Generate random data
tic                         % Start clock
y = filter(b2,a2,x);        % Filter data using the IIR filter
toc                         % Get IIR filter operation time
clear y;
tic                         % Restart clock
y = filter(b1,1,x);         % Filter using the FIR filter
toc                         % Get FIR filter operation time
```

Surprisingly, despite the differences in the number of coefficients, the two filters require around 1.2 msec for implementation. However, if `conv` is used to implement the FIR filter, then it takes three times longer.

As mentioned earlier, some well-known analog filter types can be duplicated as IIR filters. Specifically, analog filters known as Butterworth, Chebyshev types I and II, and Elliptic (or Cauer) designs can be implemented as IIR digital filters and are supported by the MATLAB Signal Processing Toolbox. Butterworth filters provide a frequency response that is maximally flat in the passband and monotonic overall. To achieve this characteristic, Butterworth filters sacrifice rolloff steepness; hence, the Butterworth filter has a less sharp initial attenuation characteristic than other filters. The other filters achieve a faster rolloff than Butterworth filters, but have ripple in the passband. The Chebyshev type II filter has ripple only in the stopband, and its passband is monotonic, but it does not rolloff as sharply as type I which has ripple in the passband. The ripple produced by Chebyshev filters is termed equiripple since it is of constant amplitude across all frequencies. Finally, Elliptic filters have steeper

rolloff than any of the above-mentioned filters, but have equiripple in both the passband and stopband. Although the sharper initial rolloff is a desirable feature as it provides a more definitive boundary between passband and stopband, most biomedical engineering applications require a smooth passband. This makes the Butterworth the filter of choice and the most commonly used in practice.

The filter coefficients for a Butterworth IIR filter can be determined using the MATLAB routine:

```
[b,a] = butter(order,wn,'ftype') ;   % Design Butterworth filter
```

where `order` and `wn` are the order and cutoff frequencies, respectively. (Of course `wn` is relative to $f_s/2$.) The other arguments are similar to those in the FIR filter `fir1`. For a low-pass filter, `wn` is scalar and the `'ftype'` argument is missing. For a band-pass filter, `wn` is a two-element vector, `[w1 w2]`, where `w1` is the low cutoff frequency and `w2` is the high cutoff frequency and again there is no `'ftype'`. If a high-pass filter is desired, then `wn` should be scalar and `'ftype'` should be `'high.'` For a stopband filter, `wn` is a two-element vector indicating the frequency ranges of the stop band and `'ftype'` should be `'stop'`. The outputs of `butter` are the `b` and `a` coefficients.

Although the Butterworth filter is the only IIR filter you are likely to use, the other filters are easily designed using the associated MATLAB routine. The Chebyshev type I and II filters are designed with similar routines except that an additional parameter is needed to specify the allowable ripple:

```
[b,a] = cheby1(order,rp,wn,'ftype');   % Design Chebyshev Type I
```

where the arguments are the same as in `butter`, except for the additional argument, `rp`, which specifies the maximum desired passband ripple in dB. The type II Chebyshev filter is designed using:

```
[b,a] = cheby2(n,rs, wn,'ftype');   % Design Chebyshev Type II
```

where again the arguments are the same, except `rs` specifies the stopband ripple, again in dB, but with respect to the passband gain. In other words, a value of 40 dB means that the ripple will not exceed −40 dB where the passband gain is 0.0 dB. In effect, this value specifies the minimum attenuation in the stopband.

The Elliptic filter includes both stopband and passband ripple values:

```
[b,a] = ellip(n,rp,rs,wn,'ftype');   % Design Elliptic filter
```

where the arguments presented are in the same manner as described earlier, with `rp` specifying the passband gain in dB and `rs` specifying the stopband ripple relative to the passband gain.

The following example uses these routines to compare the frequency response of the four IIR filters discussed earlier.

EXAMPLE 8.11

Plot the frequency response curves (in dB) obtained from an eighth-order low-pass filter using the Butterworth, Chebyshev types I and II, and Elliptic filters. Use a cutoff frequency of 200 Hz and assume a sampling frequency of 2 kHz. For all filters, the ripple or maximum attenuation should be less than 3 dB in the passband, and the stopband attenuation should be at least 60 dB.

Solution: Use the MATLAB IIR design routines to determine the a and b coefficients, use `freqz` to calculate the complex frequency spectrum, take the absolute value of the spectra and convert to dB (`20*log(abs(H))`), then plot using `semilogx`. Repeat this procedure for the four filters.

```
% Example 8.11 Frequency response of four IIR 8th-order lowpass filters
%
fs = 2000;                      % Sampling filter
n = 8;                          % Filter order
wn = 200/1000;                  % Filter cutoff frequency
rp = 3;                         % Maximum passband ripple
rs = 60;                        % Maximum stopband ripple
% Determine filter coefficients
[b,a] = butter(n,wn);           % Butterworth filter coefficients
[H,f] = freqz(b,a,256,fs);      % Calculate complex spectrum
H = 20*log10(abs(H));           % Convert to magnitude in dB
subplot(2,2,1);
semilogx(f,H,'k');              % Plot spectrum in dB vs log freq.
  .......labels and title.......
  ...... repeat for the other 3 IIR filters.......
[b,a] = cheby1(n,rp,wn);        % Chebyshev Type I filter coefficients
[b,a] = cheby2(n,rs,wn);        % Chebyshev Type II filter coefficients
[b,a] = ellip(n,rp,rs,wn);      % Elliptic filter coefficients
  .......use freqz, plot, labels, and titles.......
```

The spectra of the four filters are shown in Figure 8.24. As described earlier, the Butterworth is the only filter that has smooth frequency characteristics in both the passband and stopband; it is this feature that makes it popular in biomedical signal processing both in its analog and digital incarnations. The Chebyshev type II filter also has a smooth passband and a slightly steeper initial slope than the Butterworth, but it does have ripple in the stopband, which can be problematic in some situations. The Chebyshev type I has an even sharper initial slope, but also has ripple in the passband, which limits its usefulness. The sharpest initial slope is provided by the Elliptic filter, but

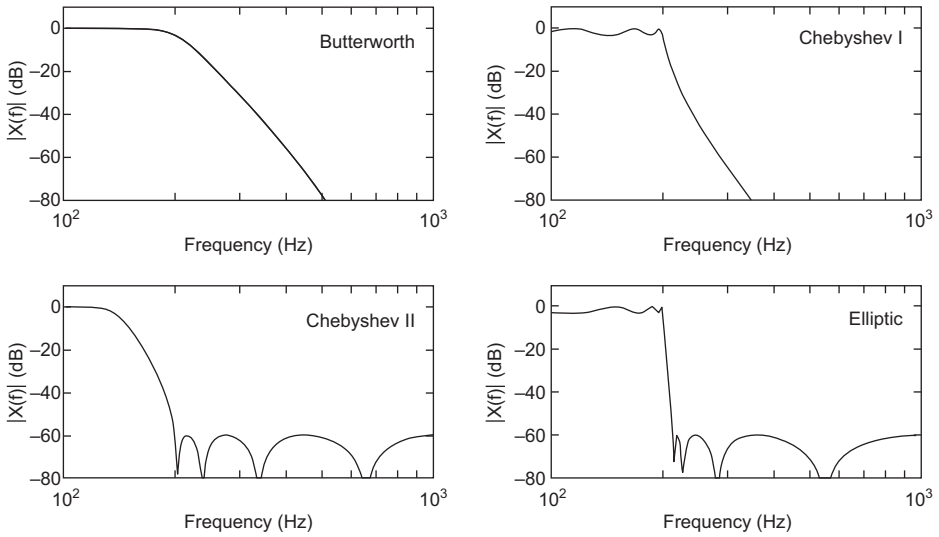

FIGURE 8.24 The spectral characteristics of four different eighth-order infinite impulse response (IIR) low-pass filters with a cutoff frequency of 200 Hz. The assumed sampling frequency is 2 kHz. It is possible to increase the initial sharpness of an IIR filter if various types of ripple in the spectral curve can be tolerated, not usually the case in bioengineering applications.

ripple is found in both the passband and stopband. Reducing the ripple of these last three filters is possible through the design routine, but this also reduces the filter's initial sharpness, yet another engineering compromise.

Other types of filters exist in both FIR and IIR forms; however, the filters described here are the most common and are the most useful in bioengineering. Given the seeming superiority of IIR filters, you might wonder why we even need FIR filters. There are two important applications where IIR do not work: image filtering and adaptive filtering. The former are described in Chapter 11, whereas the latter are left for more advanced signal processing textbooks (see Semmlow and Griffel, 2014).

8.7 SUMMARY

A digital version of the Laplace transform, called the z-transform, can be used to analyze discrete-time systems. These systems are sometimes abbreviated as LTID, which stands for linear time-invariant discrete systems. The z-transform, so-called because it uses the complex variable z in place of the Laplace variable, s, can be used to construct discrete transfer functions, $H(z)$. These transfer functions consist of numerator and denominator polynomials, just like Laplace transfer functions, except that these polynomials feature powers of z rather than powers of s (Equation 8.13). The z transfer function can be used to find the output to any input through a process similar to that used with the Laplace transfer function: take the

z-transform of the input signal, multiply it by $H(z)$, then take the inverse z-transform of the result. The z-transform and its inverse are found using tables as with the Laplace transform. This approach can be tedious and does not lend itself to computer analysis.

If the signals are assumed to be in steady state and the system has zero initial conditions, the sinusoidal variable, $e^{-j\omega}$, can be substituted for z. With this substitution, the spectrum of the system can be obtained from $H(z)$ simply by taking the Fourier transform of the numerator divided by the Fourier transform of the denominator (Equation 8.15). This operation can be done on a computer. The only discrete systems we biomedical engineers are likely to encounter are digital filters where we can assume a steady state with zero initial conditions. Hence, our interest in the z transfer function is limited to its ability to provide the system's spectrum through the Fourier transform.

Discrete systems, specifically digital filters, can be implemented using difference equations. Difference equations are discrete versions of differential equations and use the time-delay feature for the z variable. These equations consist of two terms: convolution between the signal and filter coefficients and convolution between the delayed output and additional filter coefficients (Equation 8.21). Difference equations are easy to solve using a computer and several MATLAB routines implement digital filters by solving these equations.

Filters are used to alter the spectrum of a signal, usually to eliminate frequency ranges that include noise or to enhance frequencies of interest. Filters vary in frequency range (i.e., bandwidth), basic type (low pass, high pass, band pass, and band stop), and characteristics of attenuations (rolloff and initial sharpness). Filters that have very sharp cutoffs can have ripple in the band-pass region, making them unsuitable for most biomedical engineering applications.

Designing a digital filter is a matter of determining the filter coefficients that give you the desired spectrum. Digital filters come in two basic versions: FIR and IIR. FIR filters are essentially moving average operations with the averages weighted by filter coefficients. The filter coefficients are identical to the system's impulse response. FIR filters are implemented using standard convolution. They have linear phase characteristics, but their spectra do not cut off as sharply as IIR filters of the same complexity. They can be applied to images where filter coefficients consist of a two-dimensional matrix (see Chapter 11). Their inherent stability makes them popular in adaptive filtering where the filter coefficients are continuously modified based on the signal. FIR filters can be designed by taking the inverse Fourier transform of an ideal filter, one with a rectangular frequency window. The inverse Fourier transform gives the desired filter's impulse response, which is identical to the filter coefficients. Unfortunately, in real applications, these filter coefficients must be truncated and this creates artifacts, including oscillations, in the frequency response. Appling a tapering window, such as the Hamming window, to the truncated coefficients reduces these artifacts. Although the filter coefficients of FIR rectangular window filters can be determined from basic concepts, MATLAB has routines that simplify their design.

IIR filters consist of two sets of filter coefficients, one applied to the input signal through standard convolution and the other to a delayed version of the output, and also using convolution. They can produce greater attenuation slopes and some types can produce very sharp initial cutoffs at the expense of some ripple in the band-pass region. An IIR filter known as the Butterworth filter produces the sharpest initial cutoff without band-pass ripple and is the most commonly used in biomedical applications. The design of IIR filters is more complicated than that of FIR filters, and is best achieved using MATLAB routines.

PROBLEMS

1. Find the magnitude and phase characteristic of a unit delay. (Hint: As shown in Figure 8.1, the transfer function of a unit delay is $H(z) = z^{-1}$. Refer to Equation 8.12 to help determine the a and b coefficients.) Compare the resultant magnitude and phase spectra with that of the time delay element analyzed in Example 7.2.
2. Find the spectrum (magnitude and phase) of the system represented by the z-transform:

$$H(z) = \frac{0.06 - 0.24z^{-1} + 0.37z^{-2} - 0.24z^{-3} + 0.06z^{-4}}{1 - 1.18z^{-1} + 1.61z^{-2} - 0.93z^{-3} + 0.78z^{-4}}$$

   For plotting purposes, assume $f_s = 500$ Hz. Be sure to pad the Fourier transforms sufficiently and, as always, plot only the nonredundant points.
3. Use the difference equation in Equation 8.21 to find the step response of the LTID system described by the z transfer function given in Problem 2. Use the same f_s as in Problem 2 and plot 0.25 sec of the response. (Hint: It is easiest to implement Equation 8.21 using the MATLAB `filter` routine.)
4. Use `sig_noise` to generate a 20-Hz sine wave in 5 dB of noise (i.e., SNR = −5 dB) and apply two moving average filters using the MATLAB `filter` routine: a 3-point moving average and a 10-point moving average. Plot the time characteristics of the two outputs. (Use `subplot` to combine the two plots). Use a data length (N) of 200 and remember that `sig_noise` assumes a sample frequency of 1 kHz.
5. Find the magnitude spectrum of an FIR filter with a coefficients of $b = [1\ 1\ 1\ 1\ 1\ ]/5$ in two ways: (a) apply the Fourier transform with padding to the filter coefficients and plot the magnitude spectrum; (b) pass white noise through the filter using `conv` and plot the magnitude spectra of the output. Since white noise has, theoretically, a flat spectrum, the spectrum of the filter's output to white noise should be the spectrum of the filter. In the second method, use a 20,000-point noise array, i.e., `y = conv (b, randn(20000,1))`. Use the Welch averaging method described in Section 4.4 to smooth the spectrum. For the Welch method, use a suggested segment size of 128 points and a 50% segment overlap. Since the `welch` routine (Example 4.5) produces the power spectrum, you need to take the square root to get the magnitude spectrum for comparison with the Fourier transform method. The two methods use different scaling, so the vertical axes are slightly different. (Use `subplot` to combine the two spectral plots). Assume a sampling frequency of 200 Hz for plotting the spectra.
6. Use sig_noise to construct a 512-point array consisting of two closely spaced sinusoids of 200 and 230 Hz with an SNR of −14 dB. Plot the magnitude spectrum using the Fourier transform. Generate a 24th-order (i.e., 25 coefficients) rectangular window band-pass filter using Equation 8.34 to modify the approach in Example 8.4. Set the low cutoff frequency to 180 Hz and the high cutoff frequency to 250 Hz. Apply a Blackman window (Equation 8.32, and Example 8.5) to the filter coefficients, then filter the data using MATLAB's `filter` routine. Plot the magnitude spectra before and after filtering. (Use `subplot` to combine the two spectral plots). (Hint: You can modify a section of the code in Example 8.4 to generate the band-pass filter.)

7. Write a program using Equation 8.28 to construct the coefficients of a 15th-order low-pass rectangular window filter (i.e., 16 coefficients). Assume $f_s = 1000$ and make the cutoff frequency 200 Hz. Apply the appropriate Hamming (Equation 8.31) and Blackman (Equation 8.32) windows to the filter coefficient. Find and plot the spectra of the filter without a window and with the two windows. Plot the spectra superimposed with different line types to aid comparison and pad the coefficients to $N = 256$ when determining the spectra. As always, do not plot redundant points. You can use the `blackman` routine for the Blackman window, but you need to write your own code for the Hamming window. (Suggestion: Simply modify the Blackman routine appropriately.)
8. Comparison of Blackman and Hamming windows. Generate the filter coefficients of a 127th-order low-pass rectangular window filter. Apply the Blackman and Hamming windows to the coefficients. Apply these filters to an impulse function using the MATLAB `filter` routine. The impulse input should consist of a 1 followed by 255 zeros. The impulse responses will look nearly identical, so take the Fourier transform of each the two responses and plot the magnitude and phase. You need to use MATLAB's `unwrap` routine on the phase data before plotting. (Use `subplot` to combine the magnitude and phase spectral plots). Is there any difference in the spectra of the impulses produced by the two filters? Look carefully.
9. Load file `ECG_9.mat`, which contains 9 sec of ECG data in variable `x`. These ECG data have been sampled at 250 Hz. The data have a low frequency signal superimposed over the ECG signal, possibly because of respiration artifact. Filter the data with a high-pass filter constructed using Equation 8.33. Use 65 coefficients (63rd-order) and a cutoff frequency of 8 Hz. Apply the Blackman window to the coefficients. Plot the spectrum of the filter to confirm the correct type and cutoff frequency. Also plot the filtered and unfiltered ECG data using `subplot`. (Hint: You can modify a section of the code in Example 8.4 to generate the high-pass filter.)
10. ECG data are often used to determine the heart rate by measuring the time interval between the peaks that occur in each cycle known at the "R wave." This is termed the R-R interval. To determine the position of this peak accurately, ECG data are first prefiltered with a band-pass filter that enhances the QRS complex, the spike-like section of the ECG. Load file `ECG_noise.mat`, which contains 10 s of noisy ECG data in variable `ecg`. Filter the data with a 64th-order FIR band-pass filter based on Equation 8.34 to best enhance the R-wave peaks. Determine the low and high cutoff frequencies empirically. (They will both be somewhere between 2 and 30 Hz.) The sampling frequency is 250 Hz. Plot the unfiltered and filtered signals on the same plot.
11. Load the variable `x` found in `impulse_resp1.mat`. Take the derivative of these data using the two-point central difference algorithm with a skip factor of 6, implemented using the `filter` routine. Now add an additional low-pass filter using a rectangular window filter with 65 coefficients (64th-order) and a cutoff frequency 25 Hz (Equation 8.28). Apply the filter using the `conv` routine with option `'same'` to eliminate the added delay that would be induced by the filter. Plot the original time data, the result of the two-point central difference algorithm, and the low-pass filtered derivative data. (Use `subplot` to combine the three plots). Also plot the spectrum of the two-point central difference algorithm, the low-pass filter, and the combined spectrum again combining

with `subplot`. The combined spectrum can be obtained simply by multiplying the two-point central difference spectrum point by point with the low-pass filter spectrum.

12. Load the variable `x` found in `impulse_resp1.mat`. These data were sampled at 250 Hz. Use these data to compare the two-point central difference algorithm with a differencer (Equation 8.36) combined with a low-pass filter. Use a skip factor of 8 for the two-point central difference algorithm and a 54th-order rectangular window low-pass filter (Equation 8.28) with a cutoff frequency of 20 Hz. Use MATLAB's `diff.m` to produce the difference output; then, after you divide by T_s, filter this output with a low-pass filter. Note that the derivative taken this way is smooth, but has low-frequency noise.

13. Use `sig_noise` to construct a 512-point array consisting of two widely separated sinusoids: 150 and 350 Hz, both with SNR of -14 dB ($f_s = 1$ kHz). Use MATLAB's `fir2` to design a 65th-order FIR filter having a spectrum with a double band pass. The bandwidth of the two band-pass regions should be ± 10 Hz centered about the two peaks. Plot the filter's spectrum superimposed on the desired spectrum. (Hint: To plot the desired spectrum, note that based on the calling structure of `fir2`, you can plot `m` versus `f`, but you need to rescale the frequency vector, `f`, since it is in relative frequency (relative to $f_s/2$.) Also plot the signal's spectrum before and after filtering. (Use `subplot` to combine the two signal spectra plots.)

14. The file `ECG60.mat` contains an ECG signal in variable `x` that was sampled at 250 Hz and has been corrupted by 60-Hz noise. The 60-Hz noise is at a high frequency compared with the ECG signal, so it may appear as a thick line superimposed on the signal. Construct a 126th-order FIR rectangular band-stop filter with a center frequency of 60 Hz and a bandwidth of 10 Hz and apply it to the noisy signal. Implement the filter using either `filter` or `conv` with the `'same'` option, but note the time shift if you use the former. Plot the signal before and after filtering and plot the filter spectrum to ensure that the filter is correct. You should combine all three plots using `subplot`. (Hint: You can modify a portion of the code in Example 8.5 to implement the band-stop filter.)

15. We have a unique opportunity to check a MATLAB routine. Construct the filter used in Problem 10: a 64th-order FIR band-pass filter based on Equation 8.34. Make the low and high cutoff frequencies 0.1 f_s and 0.5 f_s. Apply the Blackman window to this filter. Also construct a similar filter using `fir1`. (Remember that with MATLAB filters the cutoff frequencies are based on $f_s/2$.) Take the Fourier transform of both filter coefficients and plot the magnitude spectra superimposed. Note the slight differences perhaps because `fir1` uses a Hamming window. (Suggestion: Make $N = 256$ and arbitrarily select an f_s.)

16. Comparison of causal and noncausal FIR filter implementation. Generate a 64th-order low-pass rectangular window filter using either Equation 8.28 (Blackman window) or `fir1`. Make the cutoff frequency 200 Hz. Then apply the filter to the sawtooth wave, `x`, in file `sawth.mat`. This waveform was sampled at $f_s = 1000$ Hz. Implement the filter in two ways. Use the causal routine `filter` and the noncausal routine `conv` with option `'same'`. Plot the two waveforms along with the original superimposed for comparison. Note the differences.

17. Given the advantage of a noncausal filter with regard to the time shift shown in Problem 16, why not use a noncausal filter routinely? This problem shows the downsides of noncausal FIR filtering. Generate a 33rd-order low-pass rectangular window filter

using either Equation 8.28 (Blackman window) or `fir1`. Make the cutoff frequency 100 Hz and assume $f_s = 1$ kHz. Generate an impulse function consisting of a 1 followed by 255 zeros. Now apply the filter to the impulse function in two ways: using the MATLAB `filter` routine (causal), and the `conv` routine with the `'same'` option (noncausal). The latter generates a noncausal filter since it performs symmetrical convolution. Plot the two time responses separately limiting the x axis to 0−0.05 s to better visualize the responses. Then take the Fourier transform of each output and plot the magnitude and phase in degrees. Use the MATLAB unwrap routine on the phase data before plotting.

Note the strange spectrum produced by the noncausal filter (i.e., `conv` with the `'same'` option). This is because the implementation of the noncausal filter truncates the initial portion of the impulse response.

To confirm this, rerun the program using an impulse that is delayed by 10 sample intervals (i.e.,`impulse = [zeros(1,10) 1 zeros(1,245)];`). Note that the magnitude spectra of the two filters are now the same. The phase spectrum of the noncausal filter shows less maximum phase shift with frequency as would be expected. This problem demonstrates that noncausal filters can create an artifact with the initial portion of an input signal because of the way they compensate for the time shift.

18. Differentiate the variable `x` in the file `impulse_resp1.mat` using the two-point central difference operator with a skip factor of 10. Construct another differentiator using a 16th-order least square IIR filter implemented in MATLAB's `yulewalk` routine. The filter should perform a modified differentiator operation by having a spectrum that has a constant upward slope until some frequency f_c, and then a rapid attenuation to zero. Adjust f_c to minimize noise and still maintain derivative peaks. (A relative frequency around 0.1 is a good place to start.) To maintain the proper slope, the desired gain at the 0.0 Hz should be 0.0 and the gain at $f = f_c$ should be $\frac{f_c f_s \pi}{2}$. Plot the original data and derivative for each method side by side. The derivative should be scaled for reasonable viewing. Also plot the new filter's magnitude spectrum for the value of f_c you selected. Note the cleaner response given by this new, but somewhat more complicated derivative method because it now contains a low-pass component.

19. Compare the step response of an eighth-order Butterworth filter and a 64th-order rectangular window filter, both having a cutoff frequency of 0.2 f_s. Assume a sampling frequency of 2 kHz for plotting. Use `fir1` to generate the FIR filter coefficients and implement both filters using MATLAB's `filter` routine. Use a step input of 256 samples but delay the step by 20 samples for better visualization (i.e., the step change should occur at the 20th sample). Plot the time responses of both filters. Also plot the spectra of both filters. You can combine all the plots using `subplot`. Note the oscillations induced by filtering and also note how these oscillations differ between the two filters.

20. Repeat Problem 14, but use an eighth-order Butterworth band-stop filter to remove the 60-Hz noise. Implement the filter using either `filter` or `filtfilt`, but note the time shift if you use the former. Plot the signal before and after filtering; also plot the filter spectrum to ensure the filter is correct. You can combine all three plots using `subplot`.

21. This problem demonstrates a comparison of a causal and a noncausal IIR filter implementation. Load file `Resp_noise1.mat` containing a noisy respiration signal in variable

`resp_noise1`. Assume a sample frequency of 125 Hz. Construct a 14th-order Butterworth filter with a cutoff frequency of 0.15 $f_s/2$. Filter the respiratory signal using both `filter` and `filtfilt` and plot the original and both filtered signals. Plot the signals offset on the same graph to allow for easy comparison. Finally, plot the noise-free signal found as variable `resp` in file `Resp_noise1.mat` below the other signals. Note how the original signal compares with the two filtered signals in terms of the restoration of features in the original signal and the time shift.

22. The downsides of noncausal filtering revisited. This problem is similar to Problem 16 except that it involves an IIR filter. Generate the filter coefficients of an eighth-order Butterworth filter with a cutoff frequency of 100 Hz assuming $f_s = 1$ kHz. Generate an impulse function consisting of a 1 followed by 255 zeros. Now apply the filter to the impulse function using both the MATLAB `filter` routine and the `filtfilt` routine. The latter generates a noncausal filter. Plot the two time responses separately, limiting the x axis to 0−0.05 s to better visualize the responses.

Then take the Fourier transform of each output and plot the magnitude and phase. Use the MATLAB `unwrap` routine on the phase data before plotting. Note the differences in the magnitude spectra. The noncausal filter (i.e., `filtfilt`) has ripple in the passband. Again, this is because the noncausal filter has truncated the initial portion of the impulse response.

To confirm that the artifact is due to the initial period, rerun the program using an impulse that is delayed by 20 sample intervals (i.e., `impulse = [zeros(1,20) 1 zeros(1,235)];`). Note that the magnitude spectra of the two filters are now the same. The phase spectrum of the noncausal filter shows reduced phase shift with frequency as would be expected.

23. Load the data file ensemble_data.mat. Filter the average with a 12th-order Butterworth filter. Select a cutoff frequency that removes most of the noise, but does not unduly distort the response dynamics. Implement the Butterworth filter using `filter` and plot the data before and after filtering. Implement the same filter using `filtfilt` and plot the resultant filter data. Compare the two implementations of the Butterworth filter. For this signal, the noncausal filter works well because the interesting part is not near the edges. Use MATLAB's `text` command to display the cutoff frequency on the plot containing the filtered data.

24. FIR−IIR filter comparison. Construct a 12th-order Butterworth high-pass filter with a cutoff frequency of 80 Hz assuming $f_s = 300$ Hz. Use `fir1` to construct a FIR high-pass filter having the same cutoff frequency. Plot the spectra of both filters and adjust the order of the FIR filter to approximately match the slope of the IIR filter. Compare the number of a and b coefficients in the IIR filter with the number of coefficients in the FIR filter.

25. Find the power spectrum of an LTID system four ways: (a) use white noise as the input and take the Fourier transform of the output; (b) use white noise as an input and take the Fourier transform of the autocorrelation function of the output; (c) use white noise as an input and take the Fourier transform of the cross-correlation of the output with the input, and (d) apply Equation 8.15 to the a and b coefficients. The third approach works even if the input is not white noise.

As a sample LTID system, use a fourth-order Butterworth band-pass filter with cutoff frequencies of 150 and 300 Hz. For the first three methods, use a random input

signal with 20,000 samples. Use `crosscorr` from Chapter 2 to calculate the auto- and cross-correlation and `welch` to calculate the power spectrum. For the `welch` routine, use a window of 128 points and a 50% overlap. (Owing to the large number of samples, this program may take 60 s or more to run so you may want to debug it with fewer samples initially.)

26. Write the z-transform equation for a fourth-order Butterworth high-pass filter with a relative cutoff frequency of 0.3 $f_s/2$. (Hint: Get the coefficients from MATLAB's `butter` routine.)