4

Noise Reduction and Digital Filters

4.1 Noise Reduction

In the last chapter, specifically in Problem 3.23, we showed in a roundabout way how taking the average of three successive samples generates a new signal with reduced high-frequency components. The equation for this *moving average* method is

$$y[n] = x[n]/3 + x[n-1]/3 + x[n-2]/3 \tag{4.1}$$

where $x[n]$ is the original signal and $y[n]$ is the new signal.

Intuitively, we can see that this averaging process lessens the influence of outlying samples and smooths sharp transitions. Figure 4.1b shows the application of a three-sample moving average (Equation 4.1) to a signal in Figure 4.1a. The sharp transition is smoother and the amplitude of the outlier reduced. Of course, there is no reason to limit the average to three samples, and Figure 4.1c shows how this signal is changed by averaging over 10 samples. The transition is now very smooth and just a trace of the outlier remains. This shows that averaging can be a simple, yet powerful, tool to eliminate abrupt transitions from a signal. Such transitions are associated with high-frequency components, so this moving average process can be viewed as a lowpass filter. Later in this chapter, we show how to find the exact frequency characteristics of a moving average process.

Averaging is the fundamental concept behind all of the noise reduction techniques presented in this chapter. Unlike Equation 4.1, the weights* (the multiplying coefficients) need not all be the same. In addition, the averaging weights may be applied as a moving average (as in Equation 4.1), or as a recursive moving average so that samples already averaged are gone over again. Averaging can even be applied to whole signals, a technique known as *ensemble averaging*. This latter technique is quite powerful but has special requirements: it must be possible to obtain repeated observations of the same response.

Moving averaging is a process that is often described as a filter. Filters are used to improve the SNR of a signal and are often thought of in terms of reshaping a signal's spectrum to advantage. So, while the moving average filter is a time-domain process, its effects are usually conceptualized in the frequency domain. Most noise is broadband (the broadest-band noise being white noise with a flat spectrum), and most signals are narrowband. Accordingly, it should always be possible to improve the SNR by appropriately reshaping a waveform's spectrum using a filter.

* Filter weights are also called the *weighting function*, the *filter coefficients*, or just *coefficients* and these terms are used interchangeably to mean the weighting given to a sample in the moving average approach.

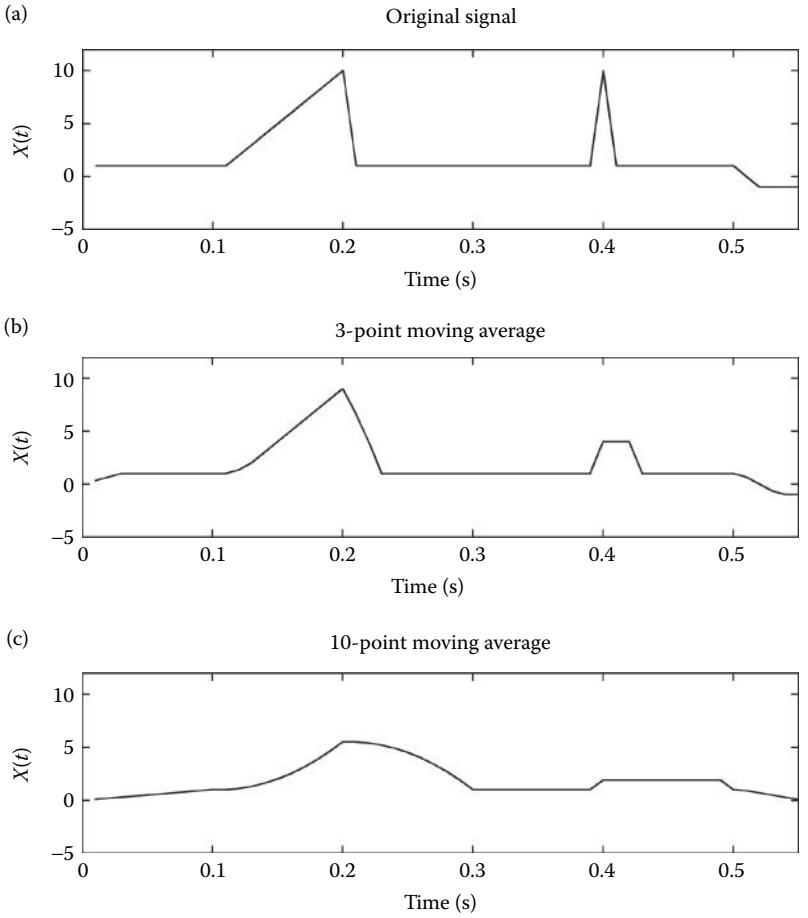

Figure 4.1 (a) An original signal segment that has a smooth ramp followed by a sharp transition at 0.12 s and an outlier at 0.4 s. (b) The signal produced by a 3-point moving average has a smoother transition and the height of the outlier reduced. (c) The signal produced by a 10-point moving average has a smooth transition and just a trace of the outlier.

A moving average can also be viewed as a linear process that acts on an input signal to produce an output signal with better (one hopes) SNR. If the filter is fixed, as are all the filters in this chapter, the filter is described as an LTI process (see Section 2.3.3). Moving average filters usually involve unequal weights (multiplying coefficients) and these weights act on the signal in the same manner as the impulse response of a linear process. Since the number of weights is finite, such moving average filters are termed *finite impulse response* (*FIR*) filters. Recursive filters apply weighted averaging to the input signal just as in FIR filters, but then they also implement a second set of weights to a delayed version of the "FIR-type" output to produce a new output. The impulse response of such recursive processes is effectively infinite, so they are termed *infinite impulse response* (*IIR*) filters. They can be particularly effective acting over only a few samples.

4.2 Noise Reduction through Ensemble Averaging

When multiple measurements are made, multiple values or signals are generated. If these measurements are combined or added together, the means add, so the combined value or signal has

a mean that is the average of the individual means. The same is true for the variance: the variances add, and the average variance of the combined measurement is the mean of the individual variances:

$$\bar{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N}\sigma_n^2 \tag{4.2}$$

When several signals are added together, it can be shown that the noise standard deviation is reduced by a factor equal to $1/\sqrt{N}$, where N is the number of measurements that are averaged.

$$\bar{\sigma}_{AVG} = \frac{\sigma}{\sqrt{N}} \tag{4.3}$$

In other words, averaging measurements from different sensors, or averaging multiple measurements from the same source, reduces the standard deviation of the measurement's variability by the square root of the number of averages. For this reason, it is common to make multiple measurements whenever possible and average the results.

In ensemble averaging, entire signals are averaged together. Ensemble averaging is a simple yet powerful signal-processing technique for reducing noise when multiple observations of the signal are possible. Such multiple observations could come from multiple sensors, but in many biomedical applications, the multiple observations come from repeated responses to the same stimulus. In ensemble averaging, several time responses are averaged over corresponding points in time. A new signal with a lower standard deviation is constructed by averaging over all signals in the ensemble. A classic biomedical engineering example of the application of ensemble averaging is the *visual-evoked response* (*VER*) in which a visual stimulus produces a small neural signal embedded in the EEG. Usually this signal cannot be detected in the EEG signal, but by averaging hundreds of observations of the EEG, time-locked to the visual stimulus, the visually evoked signal emerges.

There are two essential requirements for the application of ensemble averaging for noise reduction: the ability to obtain multiple observations, and a reference signal closely time-linked to the response. The reference signal shows how the multiple observations are to be aligned for averaging. Usually, a time signal linked to the stimulus is used.

An example of ensemble averaging is given in Example 4.1. In this example, a simulated VER data set is used. This data set not only contains the responses that would typically be recorded, but also the actual noise-free VER. Of course, it is impossible to get a noise-free version of the VER in the real world, but for our analysis here, the actual VER allows us to calculate the noise reduction produced by ensemble averaging. We can then compare this reduction with the theoretical prediction given by Equation 4.3.

EXAMPLE 4.1

This example evaluates the noise reduction provided by ensemble averaging. An ensemble of simulated VERs is found as matrix variable `ver` in MATLAB file `ver.mat`. The actual, noise-free VER signal is found as variable `actual_ver` in the same file. $T_s = 0.005$ s. Construct an ensemble average of 100 responses. Subtract out the noise-free VER from a selected individual VER and estimate the noise in each record. Then determine the noise in the ensemble average, and compare the reduction in standard deviation produced by averaging with that predicted theoretically.

Solution

Load the VER signals in file `ver.mat`. To average the signals in the matrix, we can use the MATLAB averaging routine `mean`. For a matrix, this routine produces an average of each

column, so if the signals are arranged as rows in the matrix, the routine will produce the ensemble average. To determine the orientation of the data, we check the size of the data matrix. Normally, the number of signal samples will be greater than the number of signals. Since we want the signals to be in rows, if the number of rows is greater than the number of columns, we will transpose the data using the MATLAB transposition operator (i.e., the ' symbol).

```
% Example 4.1 Example of ensemble averaging
load ver;                          % Get visual evoked response data;
Ts = 0.005;                        % Sample interval = 5 msec
[nu,N] = size(ver);                % Get data matrix size
if nu > N
     ver = ver';
     t = (1:nu)*Ts;                % Generate time vector
else
     t = (1:N)*Ts;                 % Time vector if no transpose
end
%
subplot(2,1,1);
plot(t,ver(4,:));                  % Plot individual record (Number 4)
     ....... Label axes .......
% Construct and plot the ensemble average
avg = mean(ver);                   % Take ensemble average
subplot(2,1,2);
plot(t,avg);                       % Plot ensemble average other data
     ....... Label axes .......
%
% Estimate the noise components
unaveraged_noise = ver(1,:) - actual_ver;     % Noise in individual signal
averaged_noise = avg - actual_ver;            % Noise in ensemble avg.
std_unaveraged = std(unaveraged_noise);       % Std of noise in signal
std_averaged = std(averaged_noise);           % Std of noise in ensemble avg.
theoretical = std_unaveraged/sqrt(100);       % Theoretical noise reduction
disp(' Unaveraged Averaged Theroetical')      % Output results
disp([std_unaveraged std_averaged theoretical])
```

Results

The VER is not really visible in a typical response (Figure 4.2a) but is clearly visible in the ensemble average (Figure 4.2b). The numerical output from this code is

Unaveraged Standard Deviation	Averaged Standard Deviation	Theoretical Averaged Standard Deviation
0.9796	0.0985	0.0979

Based on Equation 4.3, which states that the noise should be reduced by $\sqrt{N}$, where N is the number of responses in the average, we would expect the ensemble average to have $\sqrt{100} = 10$ times less noise than an individual record. As shown above, the standard deviation after averaging is just slightly more than the predicted value. It is not uncommon that signal-processing techniques do not reduce noise quite as much as predicted due to round-off and other errors associated with digital operations. In addition, noise is a random process, and the results of ensemble averaging are subject to some randomness. The noise reduction characteristics of ensemble averaging and its limitations are further explored in Problems 4.2 and 4.3.

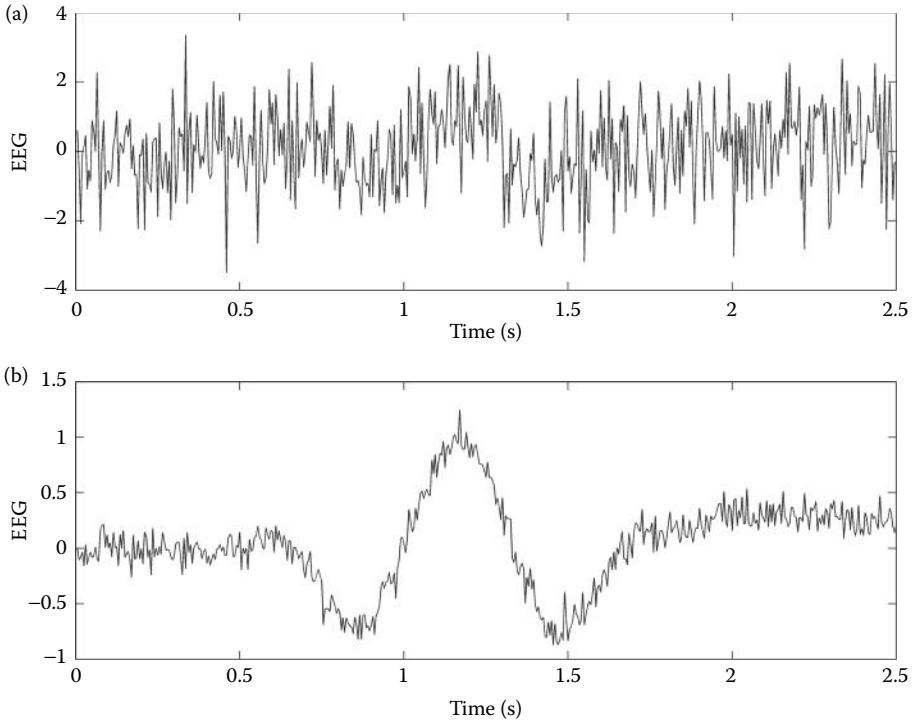

Figure 4.2 (a) The raw EEG signal showing a single response to the stimulus. (b) The ensemble average of 100 individual responses such as in graph (a) with the VER now clearly visible.

4.3 Z-Transform

The frequency-based analysis introduced in the last chapter is the most useful tool for analyzing systems or responses in which the waveforms are periodic or aperiodic, but cannot be applied to transient responses of infinite length, such as step functions, or systems with nonzero initial conditions. These shortcomings motivated the development of the *Laplace transform* in the analog domain. Laplace analysis uses the complex variable s, where $s = \sigma + j\omega$ as a representation of complex frequency in place of $j\omega$ in the Fourier transform.

$$X(\sigma,\omega) = \int_0^\infty x(t)e^{-\sigma t}\, e^{-j\omega t}dt = \int_0^\infty x(t)e^{-st}\, dt \tag{4.4}$$

The addition of the $e^{-\sigma t}$ term in the Laplace transform insures convergence for a wide range of input functions, $x(t)$, provided σ has the proper value. Like the Fourier transform, the Laplace transform is bilateral, but unlike the Fourier transform, it is not determined numerically: transformation tables and analytical calculations are used.

The *Z-transform* is a modification of the Laplace transform that is more suited to the digital domain; it is much easier to make the transformation between the discrete time domain and the Z-domain. Since we need a discrete equation, the first step in modifying the Laplace transform is to convert it to a discrete form by replacing the integral with a summation and substituting

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

$$X(\sigma,\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-\sigma n}e^{-j\omega n} = \sum_{n=-\infty}^{\infty} x[n]r^n e^{-j\omega n} \tag{4.5}$$

where $r = e^{\sigma}$.

Equation 4.5 is a valid equation for the Z-transform, but in the usual format, the equation is simplified by defining another new, complex variable. 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 $z = re^{-j\omega} = |z|\ e^{-j\omega}$, and Equation 4.5 becomes

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} = Z\left[x[n]\right] \tag{4.6}$$

This is the defining equation for the Z-transform, notated as $Z[x[n]]$. The Z-transform follows the format of the general transform equation involving projection on a basis. In this case, the basis is z^{-n}. In any real application, the limit of the summation in Equation 4.6 is finite, usually the length of $x[n]$.

As with the Laplace transform, the Z-transform is based around the complex variable, in this case, the arbitrary complex number z, which equals $|z|\ e^{j\omega}$. Like the analogous variable s in the Laplace transform, z is termed the *complex frequency*. As with the Laplace variable s, it is possible to substitute $e^{j\omega}$ for z to perform a strictly sinusoidal analysis.* This is a very useful property as it allows us to easily determine the frequency characteristic of a Z-transform function.

While the substitutions made to convert the Laplace transform to the Z-transform may seem arbitrary, they greatly simplify conversion between time and frequency in the digital domain. Unlike the Laplace transform, which requires a table and often considerable algebra, the conversion between discrete complex frequency representation and the discrete time function is easy as shown in the next section. Once we know the Z-transform of a filter, we can easily find the spectral characteristics of the filter: simply substitute $e^{j\omega}$ for z and apply standard techniques like the Fourier transform.

Equation 4.6 indicates that every data sample in the sequence $x[n]$ is associated with a unique power of z; that is, a unique value of n. The value of n defines a sample's position in the sequence. If $x[n]$ is a time sequence, the higher the value of n in the Z-transform, the further along in the time sequence a given data sample is located. In other words, in the Z-transform, the value of n indicates a time shift for the associated input waveform, $x[n]$. This time-shifting property of z^{-n} can be formally stated as

$$Z[x(n-k)] = z^{-k}Z[x(n)] \tag{4.7}$$

This time-shifting property of the Z-transform makes it very easy to implement Z-transform conversation, either from a data sequence to the Z-transform representation or vice versa.

4.3.1 Digital Transfer Function

As in Laplace transform analysis, the most useful applications of the Z-transform lies in its ability to define the digital equivalent of a transfer function. By analogy to linear system analysis, the digital transfer function is defined as

$$H[z] = \frac{Y[z]}{x[z]} \tag{4.8}$$

* If $|z|$ is set to 1, then $z = e^{j\omega}$. This is called evaluating z on the unit circle. See Smith (1997) or Bruce (2001) for a clear and detailed discussion of the properties of z and the Z-transform.

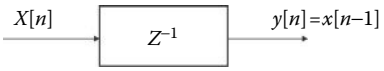

Figure 4.3 The Z-transform illustrated as a linear system. The system consists simply of a unit delay, which shifts the input by one data sample. Other powers of z can be used to provide larger shifts.

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 system known as a *unit delay* is shown in Figure 4.3. In this system, the time-shifting characteristic of z is used to define the process where the output is the same as the input, but shifted (or delayed) by one data sample. The z-transfer function for this process is just $H[z] = z^{-1}$.

Most transfer functions will be more complicated than that of Figure 4.3. They can include polynomials of z in both the numerator and denominator, just as analog transfer functions contain polynomials of s:

$$H[z] = \frac{b[0] + b[1]z^{-1} + b[2]z^{-2} + \cdots + b[K]z^{-K}}{1 + a[1]z^{-1} + a[2]z^{-2} + \cdots + a[L]z^{-L}} \tag{4.9}$$

where the $b[k]$s are constant coefficients of the numerator and the $a[\ell]$s are coefficients of the denominator.* While $H[z]$ has a structure similar to the Laplace-domain transfer function $H[s]$, there is no simple relationship between them.† For example, unlike analog systems, the order of the numerator, K (Equation 4.9), 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., no zs in the denominator) are stabler than those having higher-order denominators.

Note that, just as in the Laplace transform, a linear system is completely defined by the a and b coefficients. Equation 4.9 can be more succinctly written as

$$H[z] = \frac{\sum_{k=0}^{K} b[k]z^{-k}}{\sum_{\ell=0}^{L} a[\ell]z^{-l}} \tag{4.10}$$

From the digital transfer function, $H[z]$, it is possible to determine the output given any input:

$$Y[z] = X[x]H[z] \quad y[n] = Y^{-1}[z] \tag{4.11}$$

where $Y^{-1}[z]$ is the inverse Z-transform.

While the input–output relationship can be determined from Equation 4.9 or 4.10, it is more common to use the difference equation. This is analogous to the time-domain equation, and can

* This equation can be found in a number of different formats. A few authors reverse the roles of the a and b coefficients, with as being the numerator coefficients and bs being the denominator coefficients. More commonly, the denominator coefficients are written with negative signs changing the nominal sign of the a coefficient values. Finally, it is common to start the coefficient series with $a[0]$ and $b[0]$ so that the coefficient index is the same as the power of z. This is the notation used here; it departs from that used by MATLAB where each coefficient in the series starts with 1 (i.e., $a[1]$ and $b[1]$). So $a[0]$ here equal to $a[1]$ in MATLAB.

† 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.

be obtained from Equation 4.10 by applying the time-shift interpretation (i.e., Equation 4.7) to the term z^{-n}:

$$y[n] = \sum_{k=0}^{K-1} b[k]x[n-k] - \sum_{\lambda=1}^{L} a[\ell]y[n-\lambda] \tag{4.12}$$

This equation is derived assuming that $a[0] = 1$ as specified in Equation 4.9. In addition to its application to filters, Equation 4.12 can be used to implement any linear process given the a and b coefficients. We will find it can be used to represent IIR filter later in this chapter and the autoregressive moving average (ARMA) model described in Chapter 5.

Equation 4.12 is the fundamental equation for implementation of all linear digital filters. Given the a and b weights (or coefficients), it is easy to write a code that is based on this equation. Such an exercise is unnecessary, as MATLAB has again beaten us to it. The MATLAB routine `filter` uses Equation 4.12 to implement a wide range of digital filters. This routine can also be used to realize any linear process given its Z-transform transfer function as shown in Example 4.2. The calling structure is

```
y = filter(b,a,x)
```

where `x` is the input, `y` the output, and `b` and `a` are the coefficients in Equation 4.12. If `a = 1`, all higher coefficients are of a in Equation 4.12 are zero and this equation reduces to the convolution equation (Equation 2.55); hence, this routine can also be used to perform the convolution of `b` and `x`.

Designing a digital filter is just a matter of finding coefficients, $a[\ell]$ and $b[k]$, that provide the desired spectral shaping. This design process is aided by MATLAB routines that generate the a and b coefficients that produce a desired frequency response.

In the Laplace domain, the frequency spectra of a linear system can be obtained by substituting the real frequency, $j\omega$, for the complex frequency, s, into the transfer function. This is acceptable as long as the input signals are restricted to sinusoids and only sinusoids are needed to determine a spectrum. A similar technique can be used to determine the frequency spectrum of the Z-transfer function $H(z)$ by substituting $e^{j\omega}$ for z. With this substitution, Equation 4.12 becomes

$$H[m] = \frac{\sum_{k=0}^{K} b[k]e^{-j\omega k}}{\sum_{\lambda=0}^{L} a[\ell]e^{-j\omega\lambda}} = \frac{\sum_{k=0}^{K} b(k)e^{-2\pi mn/N}}{\sum_{\lambda=0}^{L} a[\ell]e^{-2\pi mn/N}} = \frac{FT(b[k])}{FT(a[\ell])} \tag{4.13}$$

where FT indicates the Fourier transform. As with all Fourier transforms, the actual frequency can be obtained from the harmonic number m after multiplying by f_s/N or $1/(NT_s)$.

EXAMPLE 4.2

A system is defined by the digital transfer function below. Plot its frequency spectrum: magnitude in dB and phase in degrees. Also plot the time-domain response of the system to an impulse input (i.e., the system's impulse response). Assume a sampling frequency of $f_s = 1$ kHz and make $N = 512$. These are both optional: f_s is provided so that the frequency curve can be plotted in Hz instead of relative frequency, and N is chosen to be large enough to produce a smooth frequency curve. (In this example, it is actually much larger than needed for a smooth curve: $N = 128$ would be sufficient).

$$H[z] = \frac{0.2 + 0.5z^{-1}}{1 - 0.2z^{-1} + 0.8z^{-2}}$$

Solution

In filter design, we usually work from a desired frequency spectrum, determining the a and b weights that generate a digital filter to approximate the desired spectrum. In this example, we work the other way around: we are given the coefficients in terms of the digital transfer function and are asked to find the spectrum. From the transfer function, we note that $a[0] = 1$ (as required), $a[1] = 0.2$, $a[2] = 0.8$, $b[0] = 0.2$, and $b[1] = 0.5$. We apply Equation 4.13 to find $H[m]$, then convert to frequency in Hz by multiplying m by f_s/N. To find the impulse response, we generate an impulse function (1.0 followed by zeros) and determine the output by implementing Equation 4.12 through the MATLAB `filter` routine.

```
% Example 4.2 Plot the Frequency characteristics and impulse
%  response digital system given the digital transfer function
%
fs = 1000;              % Sampling frequency (given)
N = 512;                % Number of points (given)
% Define a and b coefficients based on H(z)
a = [1 —.2 .8];         % Denominator of transfer function
b = [.2 .5];            % Numerator of transfer function
%
H = fft(b,N)./fft(a,N);             % Compute H(m). Eq. 4.13
Hm = 20*log10(abs(H));              % Get magnitude in dB
Theta = (angle(H))*360/(2*pi);      %  and phase in deg.
f = (1:N)*fs/N;                     % Frequency vector for plotting
.....plot H(m)and Theta(m),label axes, turn on grid.......
%
% Compute the Impulse Response
x = [1, zeros(1,N—1)];              % Generate an impulse input
y = filter(b,a,x);                  % Apply Eq. 4.12
......plot first 60 points of x for clarity and label.....
```

The magnitude and phase characteristics of the digital transfer function given in Example 4.2 are shown in Figure 4.4. The impulse response of this transfer function is shown in Figure 4.5 and has the form of a damped sinusoid. The digital filters described in the rest of this chapter use a straightforward application of these linear system concepts. Again, the design of digital filters is only a question of determining the $a[n]$ and $b[n]$ coefficients to produce the desired frequency characteristics.

4.4 Finite Impulse Response Filters

FIR filters are moving average filters, generally with uneven weights. In FIR filters, the filter weights (coefficients) define the impulse response, so of course the impulse response will be finite. FIR filters are implemented by convolving the weights with the input signal, which is mathematically the same as taking a running weighted average over the input signal. So, the general equation for the implementation of an FIR filter is the convolution equation presented in Chapter 2 (Equation 2.55) and repeated here with MATLAB-type indexing (i.e., from 1 to N instead of 0 to $N - 1$):

$$y[n] = \sum_{k=1}^{K-1} b[k]x[n-k] \tag{4.14}$$

where $b[k]$ defines the weights (recall, these weights are also called the *filter coefficients* or simply coefficients, the *weighting function*, or *impulse response*), K is number of weights or filter length,

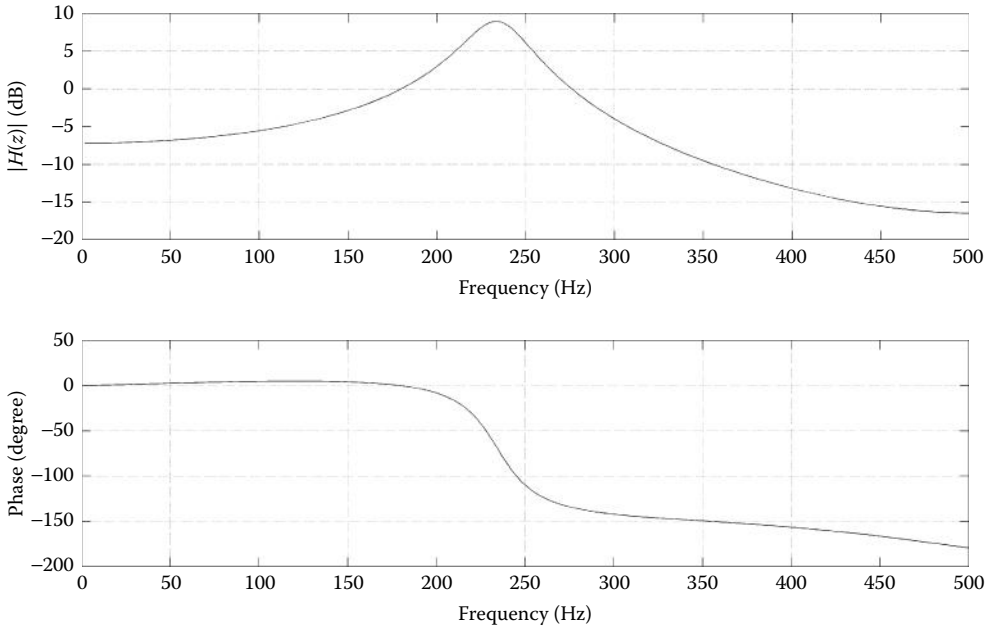

Figure 4.4 The frequency characteristic (magnitude and phase) of the digital transfer function given in Example 4.2.

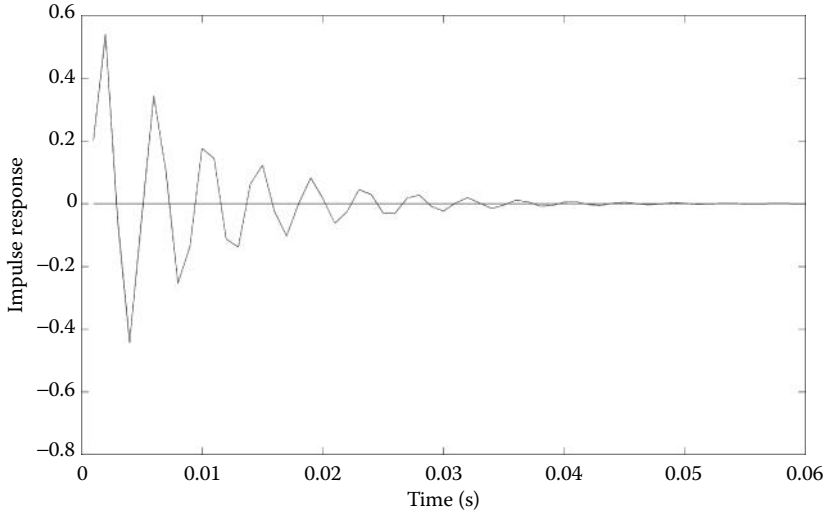

Figure 4.5 Impulse response of the system represented by the digital transfer function given in Example 4.2.

$x[n]$ is the input, and $y[n]$ is the output. FIR filters can be implemented in MATLAB using the `conv` routine. Equation 4.14 is a specific case of Equation 4.12, in which there are no a coefficients except $a[0] = 1$ (recall 0 has an index of 1 in MATLAB). So, FIR filters can also be implemented using the `filter` routine where the a coefficients are set to a value of 1.0:

```
y = filter(b,1,x);     % Implementation of an FIR filter.
```

The spectral characteristics of an FIR filter can be found from Equation 4.13. Substituting $a = a[0] = 1.0$, Equation 4.13 reduces to

$$X[m] = \sum_{k=0}^{K-1} b[k]e^{-j2\pi mn/N} = FT(b[k]) \tag{4.15}$$

This equation is supported by linear systems theory that states that the spectrum of a linear (LTI) system is the Fourier transform of the impulse response.

In the moving average FIR filter defined by Equation 4.14, each sample in the output is the average of the last N points in the input. Such a filter, using only current and past data, is termed a *causal* filter. These filters follow the cause-and-effect principle in that the output is a function of past, and perhaps present, inputs. All real-time systems are 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 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 with respect to a given sample in the output signal. 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 4.6. The upper curve in Figure 4.6a 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 data are corrupted by 60-Hz noise riding on top of the signal, a ubiquitous problem in the acquisition of biological signals. A simple 10-point, equal-weight, moving average filter was applied to the noisy data and this filter did a good job of removing the noise as shown in Figure 4.6a (lower two curves). The 10-point, moving average filter was applied in two ways: as a causal filter using only current and past values of the eye-movement data, lower curve (Figure 4.6a), and as a noncausal filter applying the filter weights to

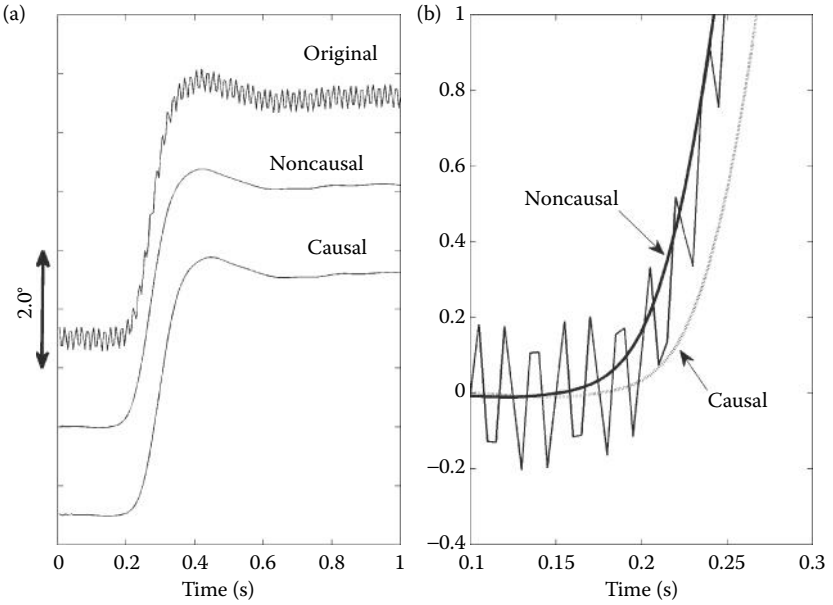

Figure 4.6 (a) Eye-movement data containing 60 Hz noise and the same response filtered with a 10-point, equal-weight, moving average filter applied in a causal and noncausal approach. (b) Detail of the initial response of the eye movement showing the causal and noncausal filtered data superimposed. The noncausal filter overlays the original data while the causal filter produces a time shift in the response.

an approximately equal number of past and future values, middle curve (Figure 4.6a). The causal filter was implemented using MATLAB's `conv` routine (i.e., Equation 4.14), with the noncausal filter using the `conv` routine with the option `'same'`:

```
y = conv(x,b,'same'); % Apply a noncausal filter.
```

where `x` is the noisy eye-movement signal and `b` is an impulse response consisting of 10 equal value samples (`b = [1 1 1 1 1 1 1 1 1 1]/10`) to create a moving average. When the `'same'` option is invoked, the convolution algorithm returns only the center section of output, effectively shifting future values into the present. The equation for a noncausal filter is similar to Equation 4.14, but with a shift in the index of x.

$$y[n] = \sum_{k=1}^{K} b[k]x[n - k + K/2] \tag{4.16}$$

Both filters do a good job of reducing the 60-cycle noise, but the causal filter has a slight delay. In Figure 4.6b, 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 4.6, 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 a noncausal filter is adequate.

The noncausal implementation of a three-point moving average filter is shown in Figure 4.7. This filter takes one past and one current sample from the input along with one future sample to construct the output sample. In Figure 4.7, a slightly heavier arrow indicates a future sample. This operation can be implemented in MATLAB using an impulse function of b = [1/3 1/3 1/3] and the `conv` routine with option `'same'`.

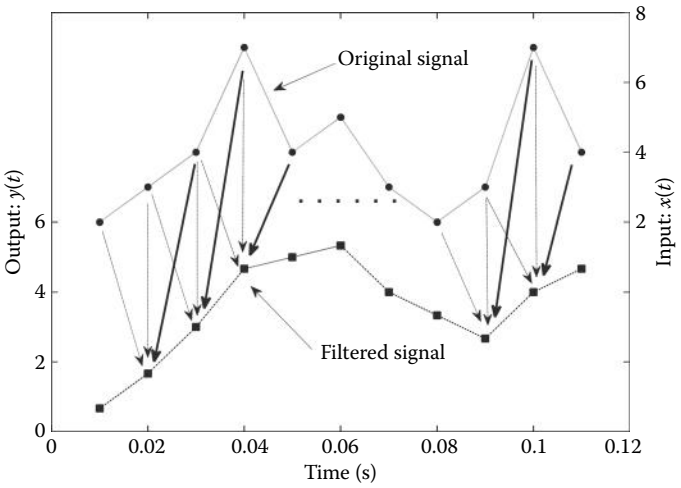

Figure 4.7 The procedure for constructing a three-point moving average. The average of each set of three points from the input makes up the output point. One point is taken from the past, one from the present, and one from the future (darker arrow). This averaging process slides across the input signal to produce the output. Endpoints are usually treated by applying zero-padding to the input signal. This process is identical to convolving the input with an impulse response of b = [1/3 1/3 1/3] in conjunction with symmetrical convolution (`conv` with option `'same'`). The time axis is scaled assuming a sampling frequency of 100 Hz.

EXAMPLE 4.3

Find the magnitude and phase spectra of a 3-point moving average filter and a 10-point moving average filter, both with equal weights. Plot the two spectra superimposed. Assume a sampling frequency of 250 Hz.

Solution

The frequency spectrum of any FIR filter can be determined by taking the Fourier transform of its filter coefficients, which is also the impulse response (Equation 4.15). However, since the impulse responses of many FIR filters can be quite short (only three points for a three-point moving average filter), it is appropriate to pad the response before taking the Fourier transform. In this example, we first construct the two filter impulse responses, then take the Fourier transform using `fft`, but pad the data out to 256 points to get smooth plots. We then plot the valid points of the magnitude and phase from the complex Fourier transform using a frequency vector based on the 250 Hz sampling rate.

```
% Example 4.3 Spectra of moving-average FIR filter
%
fs = 250;                                   % Sampling frequency
% Construct moving average filters
b3 = [1 1 1]/3;
b10 = [1 1 1 1 1 1 1 1 1 1]/10;             % Construct filter impulse response
%
B3 = fft(b3,256);                       % Find filter system spectrum
B3_mag = abs(B3);                       % Compute magnitude spectrum
Phase3 = unwrap(angle(B3));             % Compute phase angle and unwrap
Phase3 = Phase3*360/(2*pi);             % Phase angle in degrees
   .......repeat for b10 and plot the magnitude and phase spectra.......
```

Results

Figure 4.8 shows the magnitude and phase spectra of the two filters assuming a sampling frequency of 250 Hz. Both are lowpass filters with a cutoff frequency of ~40 Hz for the 3-point moving average and ~16 Hz for the 10-point moving average filter. In addition to a lower cutoff frequency, the 10-point moving average filter has a much sharper rolloff. The phase characteristics decrease linearly from 0 to ~−160 deg in a repetitive manner.

The moving average filter weights used in Example 4.3 consist of equal values; however, there is no reason to give the filter coefficients the same value. We can make up any impulse response we choose and quickly determine its spectral characteristics using the Fourier transform as in this example. The question is how do we make up filter weights that do what we want them to do in the frequency domain? This inverse operation, going from a desired frequency response to filter weights $b[k]$, is known as filter design. We could use trial and error, and sometimes that is the best approach, but in the case of FIR filter design, there are straightforward methods to get from the spectrum we want to the appropriate filter coefficients. The FIR filter design is straightforward in principle: since the frequency response of the filter is the Fourier transform of the filter coefficients (Equation 4.15), the desired impulse response is the *inverse Fourier transform* of the desired frequency response. As is often the case, the details are a little more involved, but FIR filter design is still fairly easy.

4.4.1 FIR Filter Design and Implementation

To design a filter, we start with the desired frequency characteristic. Then all we need to do is take the inverse Fourier transform. We only need the magnitude spectrum, because FIR filters

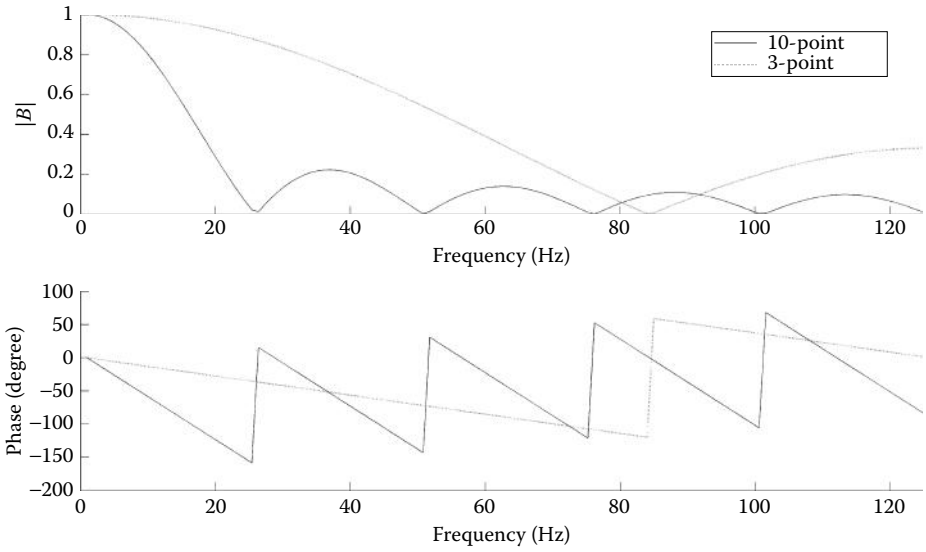

Figure 4.8 The magnitude (upper) and phase (lower) spectra of a 3-point and a 10-point moving average filter. Both filters show lowpass characteristics with a cutoff frequency, f_c, of ~40 Hz for the 3-point moving average and ~16 Hz for the 10-point moving average assuming the sampling frequency of 250 Hz. The phase characteristics change linearly and repetitively between 0 and ~–160 degree. (Note that the MATLAB's `unwrap` routine has been applied, so the jumps in the phase curve are characteristic of the filter and are not due to transitions greater than 360 degree.)

have linear phase, which is uniquely determined by the magnitude spectrum. While there may be circumstances where some specialized frequency characteristic is needed, usually we just want to separate out a desired range of frequencies from everything else and we want that separation to be as sharp as possible. In other words, we usually prefer an ideal filter such as that shown in Figure 1.9a with a particular cutoff frequency. The ideal lowpass filter in Figure 1.9a has the frequency characteristics of a rectangular window and is sometimes referred to as a *rectangular window* filter.*

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 (Equation 3.17). (This is one of those cases where we are not totally dependent on MATLAB.) When using the complex form, we must include the negative frequencies so that the desired filter's frequency characteristic is as shown in Figure 4.9. Frequency is shown in radians/s to simplify the derivation of the rectangular window impulse response. (Recall $\omega = 2\pi f$.)

To derive of the impulse response of a rectangular window filter, we apply the continuous inverse Fourier transform equation (the continuous version of Equation 3.20) to the window function in Figure 4.9. Since frequency is in radians, we use $k\omega$ for $2\pi mf$:

$$b[k] = \frac{1}{2\pi}\int_{-\pi}^{\pi} B(\omega)e^{jk\omega}d\omega = \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 1e^{jk\omega}d\omega \tag{4.17}$$

* This filter is sometimes called a *window filter*, but the term *rectangular* window filter is used in this book so as not to confuse this filter with a "window *function*" as described in Chapter 3. This can be particularly confusing since, as shown in Figure 4.9, rectangular window filters use window functions!

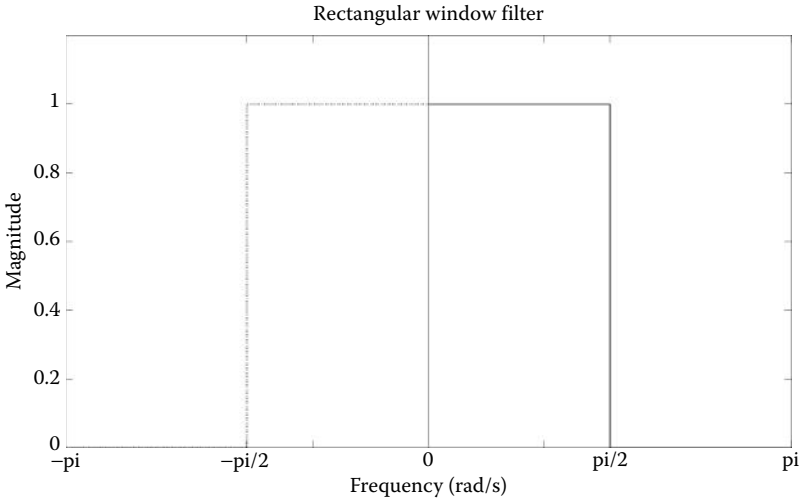

Figure 4.9 The magnitude spectrum of a rectangular window filter is an ideal filter. The negative frequency portion of this filter is also shown because it is needed in the computation of the complex inverse Fourier transform. The frequency axis is in radians per second and normalized to a maximum value of 1.0.

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

$$b[k] = \frac{1}{2\pi}\frac{e^{jk\omega}}{j}\Bigg|_{-\omega_c}^{\omega_c} = \frac{1}{k\pi}\frac{e^{jk\omega_c} - e^{-jk\omega_c}}{2j} \tag{4.18}$$

The term $e^{jk\omega_c} - e^{-jk\omega_c}/2j$ is the exponential definition of the sine function and equals $\sin(k\omega)$. So, the impulse response of a rectangular window is

$$b[k] = \frac{\sin(k\omega_c)}{\pi k} = \frac{\sin(2\pi f_c k)}{\pi k} \qquad -(L-1)/2 \le k \le (L-1)/2 \tag{4.19}$$

where the filter length, L, the number of filter coefficients must be an odd number to make $b[k]$ symmetrical about $k = 0$. When $k = 0$, the denominator goes to zero and the actual value of $b[0]$ can be obtained by applying the limits and noting that $\sin(x) \to x$ as x becomes small:

$$b[0] = \lim_{k\to 0}\left|\frac{\sin(2\pi k f_c)}{\pi k}\right| = \lim_{k\to 0}\left|\frac{2\pi k f_c}{\pi k}\right| = 2f_c \tag{4.20}$$

Substituting $\omega = f/2\omega$ into Equation 4.20 gives the value of $b[0]$ in radians/s:

$$b[0] = 2f_c = \frac{2\omega_c}{2\pi} = \frac{\omega_c}{\pi} \tag{4.21}$$

The impulse response of a rectangular window filter has the general form of a *sinc* function: $\text{sinc}(x) = \sin(x)/x$. The impulse response, $b[k]$, produced by Equation 4.19 is shown for 65 samples (k ranges between ±32) and two values of f_c in Figure 4.10. These cutoff frequencies are relative to the sampling frequency as explained below.

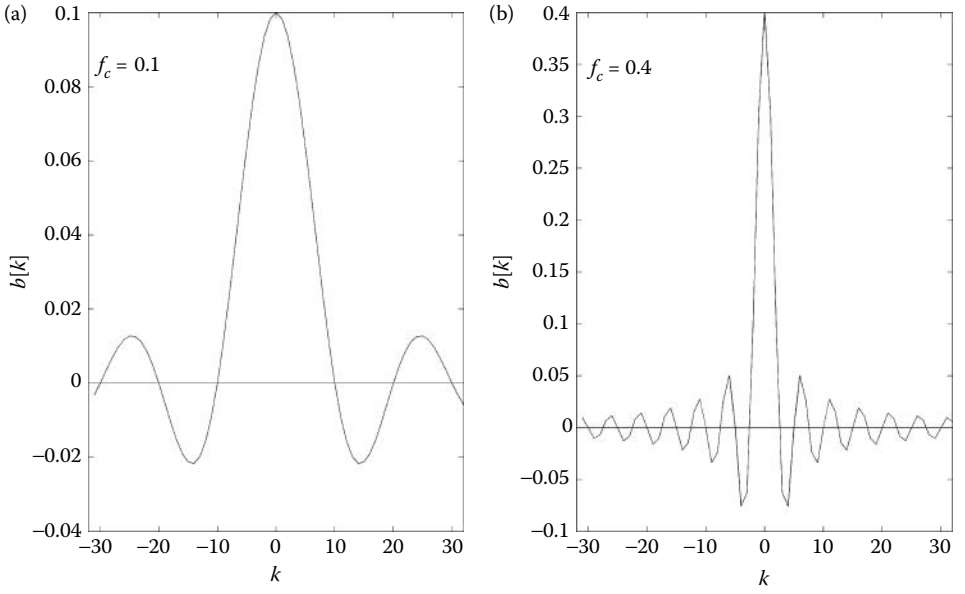

Figure 4.10 The impulse response of a rectangular window filter (Equation 4.19) for $L = 65$ coefficients (k ranges between ±32). The cutoff frequencies are given relative to the sampling frequency, f_s, as is common for digital filter frequencies. (a) Lowpass filter with a relative cutoff frequency of 0.1. (b) Lowpass filter with a higher relative cutoff frequency of 0.4 Hz.

From Figure 4.9, we can see that the cutoff frequency, ω_c, is relative to a maximum frequency of π. Converting frequency to Hz, recall that the maximum frequency in the spectrum of a sampled signal is f_s. Frequency specifications for digital filters are usually described relative to either the sampling frequency, f_s, or to half the sampling frequency $f_s/2$. (Recall that $f_s/2$ is the Nyquist frequency.) So, the cutoff frequency, f_c, in Hz is relative to the sampling frequency, f_s:

$$f_{\text{actual}} = f_c/f_s \tag{4.22}$$

In the MATLAB filter design routines included in the Signal Processing Toolbox, the cutoff frequencies are relative to $f_s/2$, the Nyquist frequency, so

$$f_{\text{actual}} = f_c/2f_s \text{ (in MATLAB filter routines)} \tag{4.23}$$

The symmetrical impulse response shown in Figure 4.10 has both positive and negative values of k. Since MATLAB requires indexes to be positive, we need to shift the output index, k, by $L/2 + 1$. If L is mistakenly given as an even number, it is adjusted so that $(L - 1)/2$ is an integer as shown in the next example.

Since we are designing FIR filters with a finite number of filter coefficients (i.e., L is finite), we need an impulse response that is also finite. Unfortunately, the solution to Equation 4.19 is an impulse response that is theoretically infinite: it does not go to zero for finite values of L. In practice, we simply truncate the impulse response equation (Equation 4.19) to some finite value L. You might suspect that limiting a filter's impulse response by truncation has a negative impact on the filter's spectrum. 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 4.4

Use the Fourier transform to find the magnitude spectrum of the rectangular window filter given by Equation 4.23 for two different lengths: $L = 18$ and $L = 66$. Use a cutoff frequency of 300 Hz assuming a sampling frequency of 1 kHz (i.e., a relative frequency of 0.3).

Solution

First, generate the filter's impulse response, $b[k]$, using Equation 4.19. This can be done by generating the negative half of b, adding $b[0] = 2f_c$, then adding the positive half of b by flipping the negative half. Since the requested L is even, the program rounds up to produce 19 and 67 coefficients. After calculating the impulse response, we find the spectrum by taking the Fourier transform of the response and plot the magnitude spectrum.

```
% Example 4.4 Generate two rectangular window impulse responses and
% find their magnitude spectra.
%
N = 256;                          % Number of samples for plotting
fs = 1000;                        % Sampling frequency
f = (1:N)*fs/N;                   % Frequency vector for plotting
fc = 300/fs;                      % Cutoff frequency (normalized to fs)
L = [18 66];                      % Requested filter lengths
for m = 1:2                       % Loop for the two filter lengths
     k = -floor(L(m)/2):-1;              % Construct k for negative b[k];
     b = sin(2*pi*fc*k)./(pi*k);         % Construct negative b[k]
     b = [b 2*fc, fliplr(b)];            % Rest of b
     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
```

The spectrum of this filter is shown in Figure 4.11 to have two artifacts associated with finite length. The oscillation in the magnitude spectrum is another example of the Gibbs artifact first encountered in Chapter 3; it is due to the truncation of the infinite impulse response. In addition, the slope is less steep when the filter's impulse response is shortened.

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 the Gibbs artifacts are due to truncation of an infinite function, we might reduce them if the impulse response were tapered toward zero rather than abruptly truncated. Some of the tapering window functions described in Chapter 3 can be useful in mitigating the effects of the abrupt truncation. In Chapter 3, window functions such as the Hamming window are used to improve the spectra obtained from short data sets, and FIR impulse responses are usually short. There are many different window functions, but the two most popular and most useful for FIR impulse responses are the Hamming and the Blackman–Harris windows described in Section 3.2.4. The Hamming window equation is given in Equation 3.26 and the Blackman–Harris window in Equation 3.27, both repeated here:

$$w[n] = 0.5 - 0.46\cos\left(\frac{2\pi n}{N}\right) \tag{4.24}$$

$$w[n] = a_0 + a_1\cos\left(\frac{2\pi}{N}n\right) + a_2\cos\left(\frac{2\pi}{N}2n\right) + a_3\cos\left(\frac{2\pi}{N}3n\right) \tag{4.25}$$

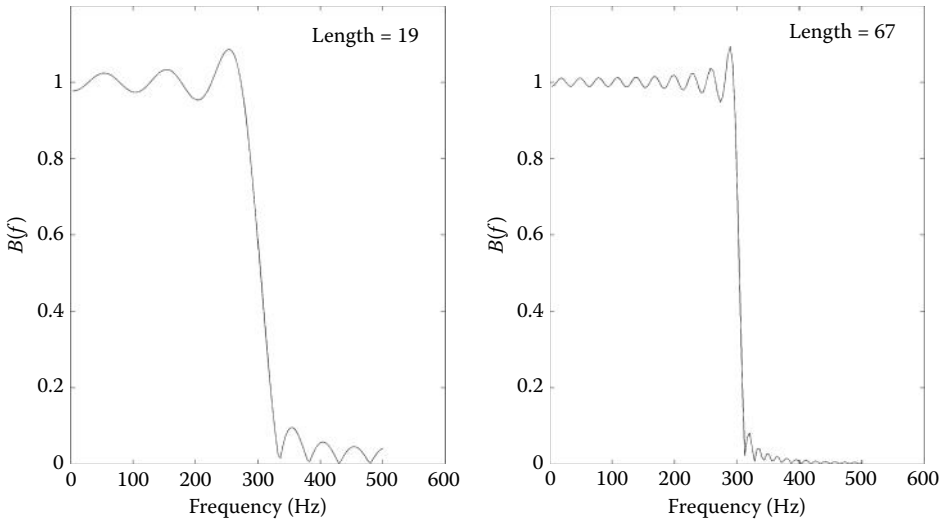

Figure 4.11 Magnitude spectra of two FIR filters based on an impulse derived from Equation 4.19. The impulse responses are abruptly truncated at 19 and 69 coefficients. The lowpass cutoff frequency is 300 Hz for both filters at an assumed sample frequency of 1 kHz. The oscillations seen are the Gibbs artifacts and are due to the abrupt truncation of what should be an infinite impulse response. Like the Gibbs artifacts seen in Chapter 3, they do not diminish with increasing filter length, but do increase in frequency.

where N is the length of the window, which should be the same length as the filter (i.e., $N = L$). Typically, $a_0 = 0.35875$; $a_1 = 0.48829$; $a_2 = 0.14128$; $a_3 = 0.01168$.

The example below applies the Blackman–Harris window the filters of Example 4.4. Coding the Hamming window is part of one of the problems at the end of this chapter.

EXAMPLE 4.5

Apply a Blackman–Harris window to the rectangular window filters used in Example 4.4. (Note that the word "window" is used in two completely different contexts in the last sentence. The first "window" is a time-domain function; the second is a frequency-domain function. Engineering terms are not always definitive.) Calculate and display the magnitude spectrum of the impulse functions after they have been "windowed."

Solution

Generate a Blackman–Harris window equivalent to the length of the filter. Use a vector of `n = -floor(L/2):floor(L/2)` to generate the window. Apply it to the filter impulse responses using point-by-point multiplication (i.e., using the .* operator). Modify the last example applying the window to the filter's impulse response before taking the Fourier transform.

```
% Example 4.5 Apply the Blackman-Harris window to the rectangular window
impulse
% responses developed in Example 4.4.
%
     .......Same code as in Example 4.4 up to.......
b = [b 2*fc, fliplr(b)];          % Rest of b
```

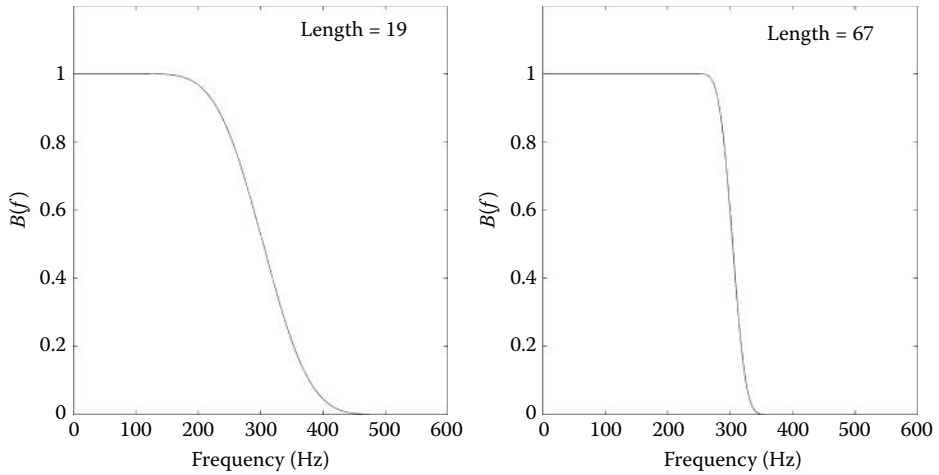

Figure 4.12 Magnitude spectrum of the FIR filters shown in Figure 4.11 except a Blackman–Harris window was applied to the filter coefficients. The Gibbs oscillations seen in Figure 4.11 are no longer visible.

```
N_w = length(b);                    % Window length
n = -floor(L(m)/2):floor(L(m)/2); % Window vector
w_B = 0.35875 + 0.48829*cos(2*pi*n/N_w) +...
     0.14128*cos(4*pi*n/N_w) + 0.01168*cos(6*pi*n/N_w);
b = b .* w_B;                       % Apply window to impulse response
H = fft(b,N);                       % Calculate spectrum
     .......same code as in Example 4.4, plot and label.......
```

The Blackman–Harris window is easy to generate in MATLAB (even easier if using the Signal Processing Toolbox window routines) and, when applied to the impulse responses of Example 4.4, substantially reduces the oscillations as shown in Figure 4.12. The filter rolloff is still not that of an ideal filter, but becomes steeper with increased 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.

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

EXAMPLE 4.6

Apply a lowpass rectangular window filter to the 10-min respiration signal shown in the top trace of Figure 4.13 which can be found as variable `resp` in file `Resp.mat`. The signal is sampled at 12.5 Hz. The low sampling frequency is used 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–Harris window to truncate the filter's impulse response. Plot the original and filtered signal.

Solution

Reuse the code in Example 4.4 and apply the filter to the respiratory signal using convolution. For variety, the Blackman–Harris filter will be generated using the Signal Processing Toolbox routine, `blackman`. MATLAB's `conv` routine is invoked using the option `'same'` to implement a causal filter and avoid a time shift in the output.

```
Example 4.6 Apply a rectangular window filter to noisy data
%
load Resp;                          % Get data
N = length(resp);                   % Get data length
fs = 12.5;                          % Sample frequency in Hz
t = (1:N)/fs;                       % Time vector for plotting
L = 65;                             % Filter lengths
fc = 1.0/fs;                        % Filter cutoff frequency: 1.0 Hz
plot(t,resp + 1,'k'); hold on;      % Plot original data, offset for clarity
k = -floor(L/2):-1;                 % Construct k for negative b[k];
b = sin(2*pi*fc*k)./(pi*k);         % Construct negative b[k]
b = [b 2*fc, fliplr(b)];            % Rest of b
b = b.*blackman(L);                 % Apply Blackman window
%
y = conv(resp,b,'same');            % Apply filter (causal)
plot(t,y);                          % Plot filtered data
```

Results

The filtered respiratory signal is shown in the lower trace of Figure 4.13. The filtered signal is smoother than the original signal as the noise riding on the original signal has been eliminated.

The FIR filter coefficients for highpass, bandpass, and bandstop filters can also be derived by applying an inverse FT to rectangular spectra having the appropriate associated shape. These equations have the same general form as Equation 4.19 except they may include additional terms:

$$b[k] = \begin{cases} \dfrac{-\sin(2\pi k f_c)}{\pi k} & k \neq 0 \\ 1 - 2f_c & k = 0 \end{cases} \quad \text{Highpass} \tag{4.26}$$

$$b[k] = \begin{cases} \dfrac{\sin(2\pi k f_h)}{\pi k} - \dfrac{\sin(2\pi k f_l)}{\pi k} & k \neq 0 \\ 2(f_h - f_l) & k = 0 \end{cases} \quad \text{Bandpass} \tag{4.27}$$

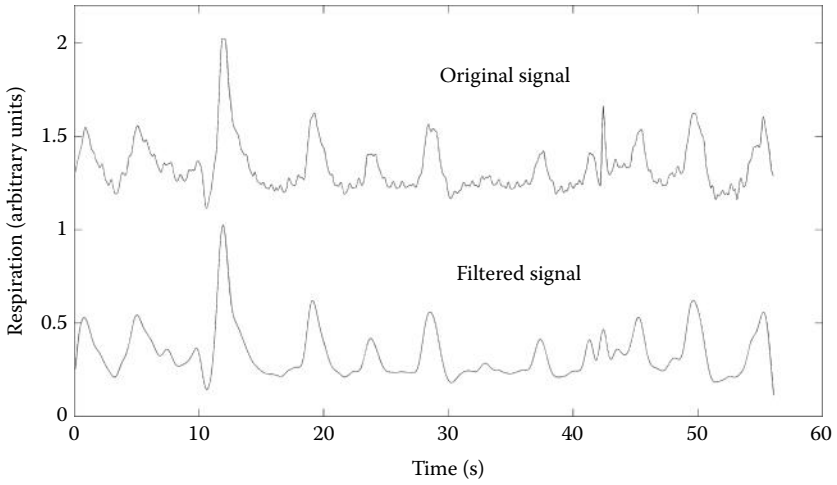

Figure 4.13 Results from Example 4.6. The output of a respiratory monitor is shown in the upper trace to have some higher-frequency noise. In the lower trace, the signal has been filtered with an FIR rectangular lowpass filter with a cutoff frequency of 1.0 Hz and a filter length of 65. (Original data from PhysioNet, Goldberger et al., 2000.)

$$b[k] = \begin{cases} \dfrac{\sin(2\pi k f_l)}{\pi k} - \dfrac{\sin(2\pi k f_h)}{\pi k} & k \neq 0 \\ 1 - 2(f_h - f_l) & k = 0 \end{cases} \quad \text{Bandstop} \tag{4.28}$$

The order of highpass and bandstop filters should always be even, so the number of coefficients in these filters should be odd. The next example applies a bandpass filter to the EEG introduced in Chapter 1.

EXAMPLE 4.7

Apply a bandpass 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–Harris window to truncate the filter's impulse to 129 coefficients. Plot the data before and after bandpass filtering. Also plot the spectrum of the original signal and superimpose the spectrum of the bandpass filter.

Solution

Construct the filter's impulse response using a bandpass equation (Equation 4.30). Use the same strategy for constructing the $b[k]$ as in the previous three examples. Note that $b[0] = 2f_h - 2f_l$. Recall that the sampling frequency of the EEG signal is 50 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 4.7 Apply a bandpass filter to the EEG data in file ECG.mat.
%
load EEG;                          % Get data
fs = 50;                           % Sample frequency
N = length(eeg);                   % Find signal length
fh = 12/fs;                        % Set bandpass cutoff frequencies
fl = 6/fs;
L = 129;                           % Set number of weights as 129
k = -floor(L/2):-1;                    % Construct k for negative b[k];
b = sin(2*pi*fh*k)./(pi*k) - sin(2*pi*fl*k)./(pi*k);       % Neg. b[k]
b = [b 2*(fh-fl), fliplr(b)];                % Rest of b[k]
b = b .* blackman(L)';                       % Apply Blackman-Harris
                                             % window to filter coefficients
y = conv(eeg,b,'same');                      % Filter using convolution
          ......plot eeg before and after filtering........
          ......plot eeg and filter spectra........
```

Bandpass 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 signal (Figure 4.14). This figure shows that filtering can significantly alter the appearance and interpretation of biomedical data. The bandpass spectrum shown in Figure 4.15 has the desired cutoff frequencies and, when compared to the EEG spectrum, is shown to reduce sharply 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 the filter 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.

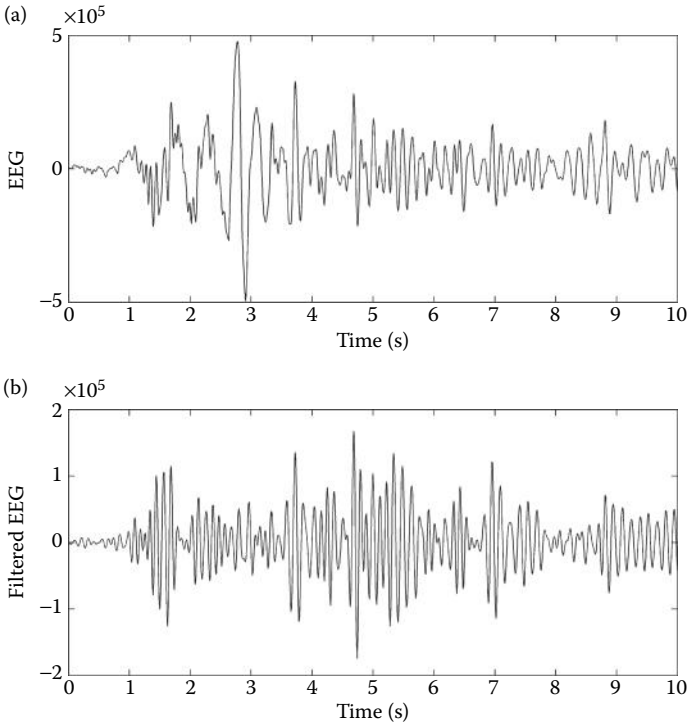

Figure 4.14 EEG signal before (a) and after (b) bandpass filtering between 6 and 12 Hz. A fairly regular oscillation is seen in the filtered signal.

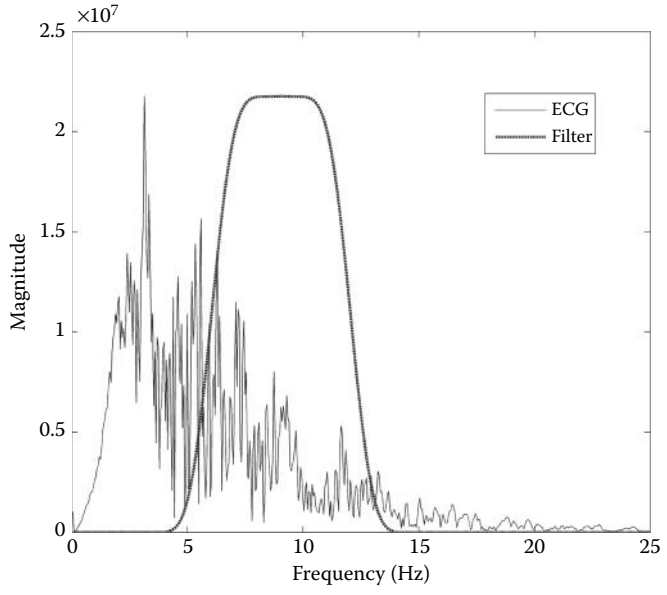

Figure 4.15 The magnitude spectrum of the EEG signal used in Example 8.5 along with the spectrum of a bandpass filter based in Equation 4.27. The bandpass filter range is designed with low and high cutoff frequencies of 6 and 12 Hz.

4.4.2 Derivative Filters: 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:

$$\frac{dx[n]}{dn} = \frac{\Delta x[n]}{T_s} = \frac{x[n+1] - x[n]}{T_s} \tag{4.29}$$

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 Figure 4.17, the frequency characteristic of the derivative operation increases linearly with frequency, so differentiation enhances higher-frequency signal components. 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 4.16a is a fairly clean physiological motor response, an eye movement similar to that shown in Figure 4.6. The lower curve of Figure 4.16a is the velocity of the movement obtained by calculating the derivative using MATLAB's `diff` routine. Considering the relative smoothness of the original signal, the velocity curve obtained using Equation 4.28 is quite noisy.

In the context of FIR filters, Equation 4.29 is equivalent to a two-coefficient impulse function, $[+1, -1]/T_s$. (Note that the positive and negative coefficients are reversed by convolution, so they must be 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 at higher frequencies that are likely to be only noise. The *two-point central difference algorithm* achieves just such an effect, acting as a differentiator at lower frequencies and a lowpass filter at higher frequencies. Figure 4.16b shows the same response and derivative 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.

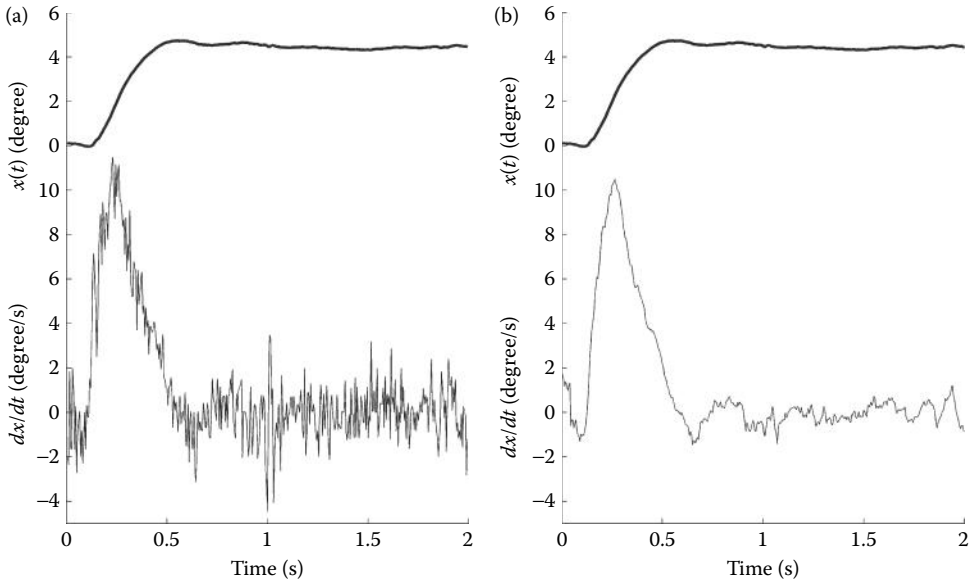

Figure 4.16 An eye-movement response to a step change in target depth is shown in the upper trace, and its velocity (i.e., derivative) is shown in the lower trace. (a) The derivative is calculated by taking the difference in adjacent points and scaling it by the sample interval (Equation 4.29). The velocity signal is noisy, even though the original signal is fairly smooth. (b) The derivative is computed using the two-point central difference algorithm (Equation 4.30) with a skip factor of 4.

The two-point central difference algorithm still subtracts two points to get a slope, but the two points are no longer adjacent; 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 this differentiator is

$$\frac{dx[n]}{dn} = \frac{x[n+L] - x[n-L]}{2LT_s} \tag{4.30}$$

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. L influences the effective bandwidth of the filter, as is shown below. Equation 4.30 can be implemented as an FIR filter using filter coefficients:

$$b[k] = \begin{cases} 1/2LT_s & k = -L \\ -1/2LT_s & k = +L \\ 0 & k \neq \pm L \end{cases} \tag{4.31}$$

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 as well as in the usual manner using MATLAB. Both methods are presented in the example below.

EXAMPLE 4.8

(a) Determine the magnitude spectrum of the two-point central difference algorithm analytically, and then (b) use MATLAB to determine the spectrum.

Analytical Solution

Starting with the equation for the discrete Fourier transform (Equation 3.18) substituting k for n in that equation:

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

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

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

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

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 $f = m/(NTs)$; hence, $m = fNT_s$. Substituting in fNT_s for m and taking the magnitude of $X[m]$, now $X(f)$:

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

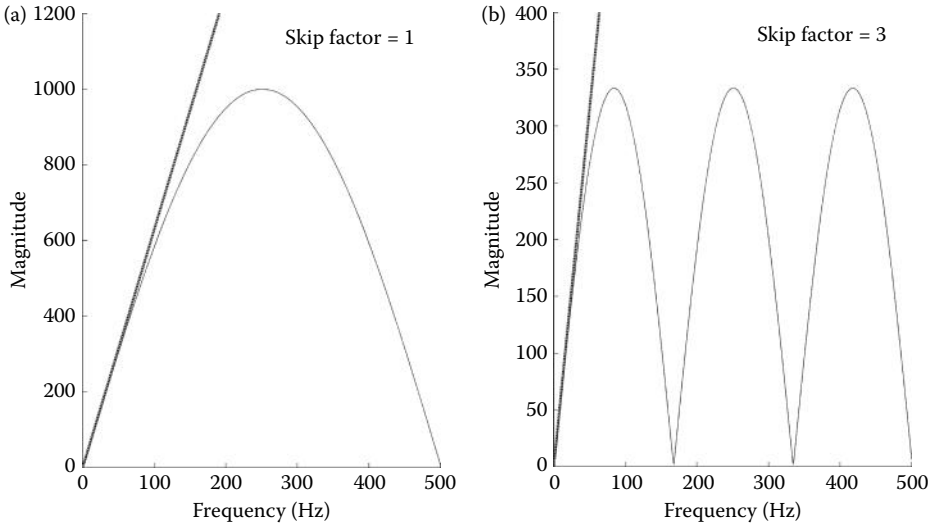

Figure 4.17 The frequency response of the two-point central difference algorithm using two different skip factors: (a) $L = 1$; (b) $L = 3$. The darker line shows the frequency characteristic of a simple differencing operation. The sample frequency is 1.0 kHz.

This equation shows that the magnitude spectrum, $|X(f)|$, is a sine function that goes to zero at $f = 1/(2LT_S) = f_s/(2L)$. Figure 4.17 shows the frequency characteristics of the two-point central difference algorithm for two different skip factors: $L = 1$ and $L = 3$.

MATLAB Solution

Finding the spectrum using MATLAB is straightforward once the impulse response 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 4.31 directly. Again, the initial filter coefficient is positive, and the final coefficient negative, to account for the reversal produced by convolution.

```
% Example 4.8 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 = 1 sec)
Ln = [1 3];                   % Define two different skip factors
for m = 1:2                   % Repeat for each skip factor
 L = Ln(m);                   % Set skip factor
 b = [1 zeros(1,2*L + 1) -1]/(2*L*Ts)];    % Filter impulse resp.
 H = abs(fft(b,N));           % Calculate magnitude spectrum
 subplot(1,2,m);                          % Plot the result
 .......plot and label spectrum; plot straight line for comparison......
end
```

The result of this program and the analytical analysis is shown in Figure 4.17. 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 4.17. The two-point central difference spectrum approximates a true derivative over the lower frequencies, but has the characteristic of a lowpass 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 lowpass filter range. Note that

for skip factors >1, the response curve repeats at $f = 1/(2LT_S)$. 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 lowpass filter as shown in one of the problems. It is also possible to combine the difference equation (Equation 4.31) with a lowpass filter; this is also explored in one of the problems.

4.4.2.1 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. In fact, there is an approach called the *Wiener filtering* that leads to an optimal filter if these frequencies are accurately known (see Chapter 8). However, this knowledge is usually not available in biomedical engineering applications. In most cases, filter parameters such as filter order (i.e., $L1$) and cutoff frequencies are set empirically based on the data. In one scenario, the signal bandwidth is progressively reduced until some desirable feature of the signal is lost or compromised. While it is not possible to establish definitive rules due to the task-dependent nature of filtering, the next example gives an idea about how these decisions are approached.

In the next example, we return to the eye-movement signal. We evaluate several different skip factors to find the one that gives the best reduction of noise without reducing the accuracy of the velocity trace.

EXAMPLE 4.9

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 it on the associated plot. The original eye movement (Figure 4.16) (upper traces) is in degrees, so the velocity trace is in degrees per second.

```
% Example 4.9 Evaluate the two-point central difference algorithm using
% different derivative skip factors
%
load eye;                          % Get data
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 m = 1:4                        % Loop for different skip factors
 b = [1 zeros(1,2*L(m)-1) -1]/(2*L(m)*Ts);  % Construct filter
 der = conv(eye_move,b,'same');            % Apply filter
 subplot(2,2,m);                   % Plot in different positions
 plot(t,der,'k');
 text(1,22,['L = ',num2str(L(m))],'FontSize',12); % Add text to plot
 text(1,18,['Peak = ',num2str(max(der),2)],'FontSize',12);
.......labels and axis.......
end
```

The results of this program are shown in Figure 4.18. As the skip factor increases the noise decreases, and the velocity trace becomes quite smooth at a skip factor of 10. However, often we

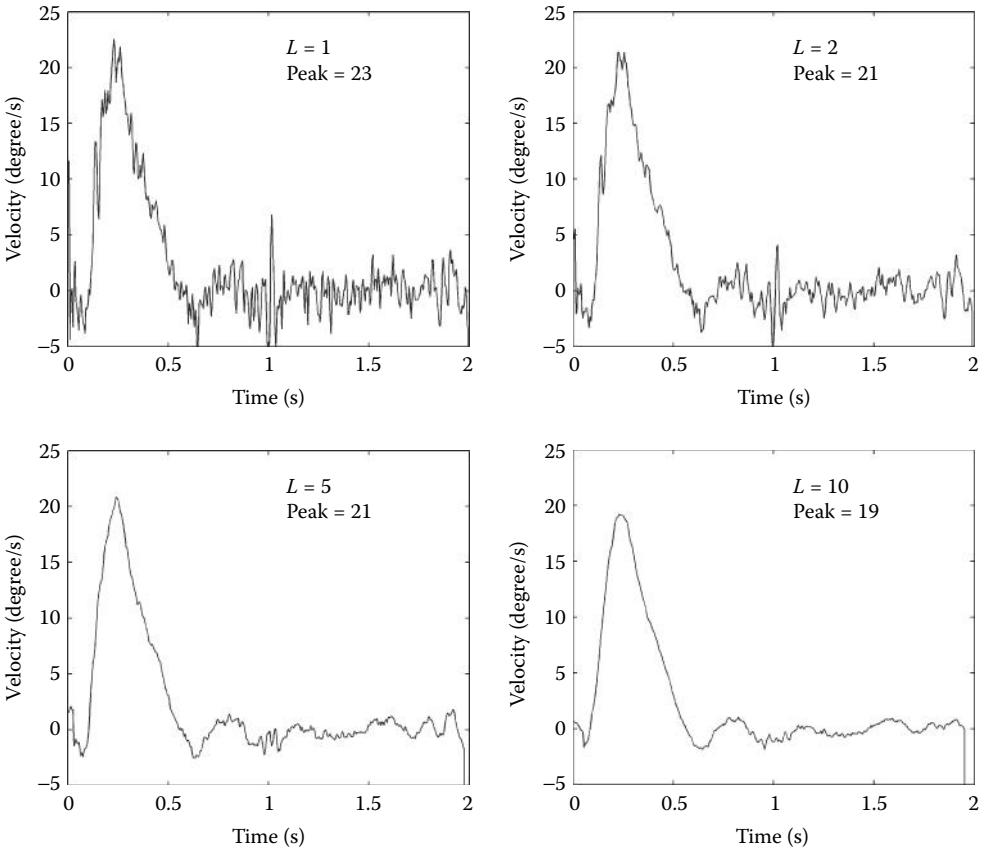

Figure 4.18 Velocity traces for the eye movement shown in Figure 4.16 (upper traces) calculated by the two-point central difference algorithm for different values of skip factor as shown. The peak velocities are shown in degrees per second.

are concerned with measuring the maximum velocity of a response, and the peak velocity also decreases with the higher skip factors. Examining the curve with the lowest skip factor ($L = 1$) suggests that the peak velocity shown, 23 deg/s, may be augmented a bit by noise. A peak velocity of 21 deg/s is found for both skip factors of two and five. This peak velocity appears to be reflective of the true peak velocity. The peak found using a skip factor of 10 is slightly reduced by the derivative filter. Based on this empirical study, a skip factor of around five seems appropriate, but this is always a judgment call. In fact, judgment and intuition are all too frequently involved in signal-processing and signal analysis tasks, a reflection that signal processing is still an art. Other examples that require empirical evaluation are given in the problem set.

4.4.3 FIR Filter Design Using MATLAB

The rectangular window equations facilitate the design of all the basic FIR filter types: lowpass, highpass, bandpass, and bandstop. In some rare cases, there may be a need for a filter with a more exotic spectrum; MATLAB offers considerable support for the design and evaluation of both FIR and IIR filters in the Signal Processing Toolbox.

Within the MATLAB environment, filter design and application occur in either one or two stages where each stage is executed by different, 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. Here, we cover only the single-stage approach, in which you specify the filter order, since trial and error is often required to determine the filter order anyway.

If a filter task is particularly demanding, the Signal Processing Toolbox provides an interactive filter design package called FDATool (for filter design and analysis tool) that uses an easy graphical user interface (GUI) to design filters with highly specific or demanding spectral characteristics. Another Signal Processing Toolbox package, the SPTool (for Signal Processing Tool), is useful for analyzing filters and generating spectra of both signals and filters. MATLAB help contains detailed information on the use of these two packages.

Irrespective of the design process, the net result is a set of b coefficients, the filter's impulse response. The FIR filter represented by these coefficients is implemented as above, using either MATLAB's `conv` or `filter` routine. Alternatively, the Signal Processing Toolbox contains the routine `filtfilt` that, like `conv` with the `'same'` option, eliminates the time shift associated with standard filtering. A comparison between `filter` and `filtfilt` is given in one of the problems.

One useful Signal Processing Toolbox routine determines the frequency response of a filter, given the coefficients. We already know how to do this using the Fourier transform, but the MATLAB routine `freqz` also includes frequency scaling and plotting so while it is not essential, it is 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. Only the `b` coefficients are associated with FIR filters, so `a` is set to 1.0. 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. If the output arguments are specified, the output vector H is the complex frequency response of the filter (the same variable produced by `fft`) and `f` is a frequency vector useful in plotting. If `fs` is given, `f` is in Hz and ranges between 0 and $f_s/2$; otherwise, `f` is in rad/sample and ranges between 0 and π.

The MATLAB Signal Processing Toolbox has a filter design routine based on the rectangular window filters described above: `fir1`. The basic rectangular window filters provided by these routines will not be explained here, but its calling structure can easily be found in `help fir1`. Like `freqz`, these routines do things we already know how to do; they are just more convenient.

A filter design algorithm that is more versatile than the rectangular window filters already described is `fir2`, which can generate a filter spectrum having an arbitrary shape. The command structure for `fir2` is

```
b = fir2(order,f,G);   % Design special FIR filter
```

where `order` is the filter order (i.e., the number of coefficients in the impulse response), `f` is a vector of normalized frequencies in ascending order, and `G` is the desired gain of the filter at the corresponding frequency in vector `f`. In other words, `plot(f,G)` shows the desired magnitude frequency curve. Clearly, `f` and `G` 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 $f_s/2$). As with all MATLAB filter design routines, frequencies are normalized to $f_s/2$ (not f_s): that is, an `f` = 1 is equivalent to the frequency

$f_s/2$. Some additional optional input arguments are mentioned in the MATLAB Help file. An example showing the flexibility of the `fir2` design routine is given next.

EXAMPLE 4.10

Design a double bandpass filter that has one passband between 50 and 100 Hz and a second passband between 200 and 250 Hz. Use a filter order of 65. Apply this to a signal containing sinusoids at 75 and 225 Hz in –20 dB of noise. (Use `sig_noise` to generate the signal). Plot the desired and actual filter magnitude spectra and the magnitude spectra of the signals before and after filtering.

Solution

We can construct a double bandpass filter by executing two bandpass filters in sequence, but we can also use `fir2` to construct a single FIR filter having the desired filter characteristics. First, we specify the desired frequency characteristics in terms of frequency and gain vectors. 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. Then, we construct the coefficients (i.e., impulse response) using `fir2`, and apply it using `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 `freqz` routine.

```
% Ex 4.10 Design a double bandpass filter.
%
fs = 1000;                      % Sample frequency
N = 2000;                       % Number of points
L = 65;                         % Filter order
fl1 = 50/(fs/2);                % Define cutoff freqs: first peak low
fh1 = 100/(fs/2);               % First peak high freq.
fl2 = 200/(fs/2);               % Second peak low cutoff
fh2 = 250/(fs/2);               % Second peak high cutoff
%
x = sig_noise([75 225],-20,N);  % Generate noisy signal
%
% Design filter Construct freqeuncy and gain vectors
f = [0,fl1,fl1,fh1,fh1,fl2,fl2,fh2,fh2,1];        % Frequency vector
G = [0, 0, 1, 1, 0, 0, 1, 1, 0, 0];               % Gain vector
subplot(2,1,1); hold on;                      % Set up to plot spectra
plot(f,G);                                    % Plot the desired response
        .......labels.......
b = fir2(L,f,G);                              % Construct filter
[H,f] = freqz(b,1,512,fs);                    % Calculate filter response
subplot(2,1,2);
plot(f,abs(H));                               % Plot filter freq. response
........labels......
y = filter(b,1,x);              % Apply filter
Xf = abs(fft(x));               % Compute magnitude spectra of filtered
Yf = abs(fft(y));               % and unfiltered data
 ........plot and label magnitude spectra of data.......
```

Figure 4.19a is a plot of the desired magnitude spectrum obtained simply by plotting the gain vector against the frequency vector (`plot(f,A)`). Figure 4.19b shows the actual magnitude spectrum that is obtained by the `freqz` routine; this could have been found using the Fourier transform using Equation 4.13. The effect of applying the filter to the noisy data is shown by the magnitude spectra in Figure 4.20. The spectrum of the unfiltered data (Figure 4.20a) shows the

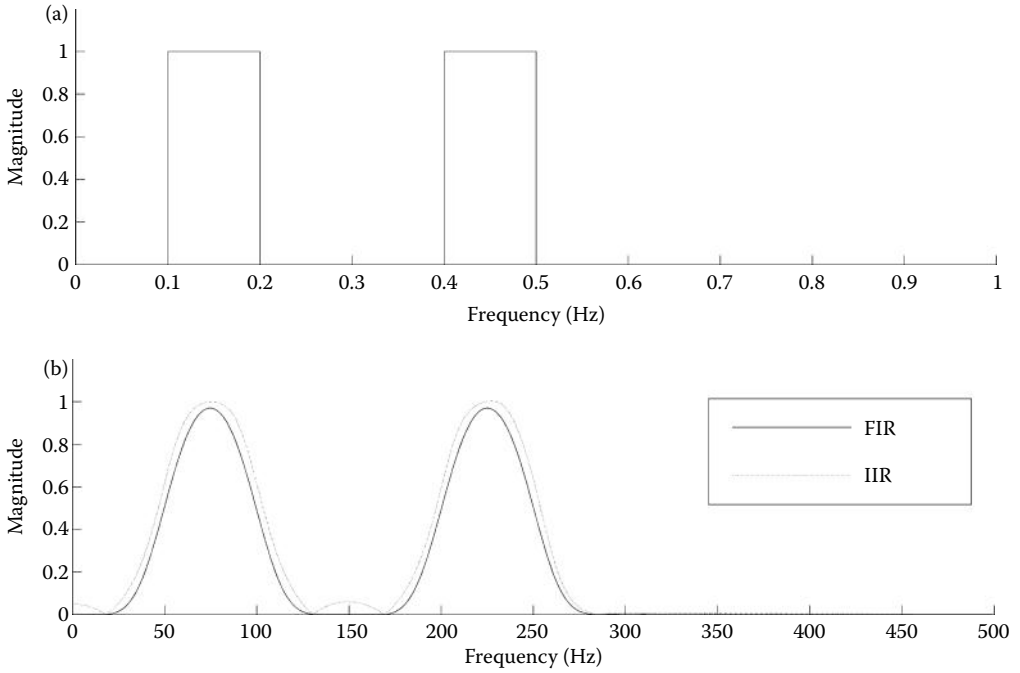

Figure 4.19 (a) Desired magnitude response for the double bandpass filter in Example 4.10. (b) Actual magnitude spectrum obtained from a 65th-order (i.e., 65 coefficients) FIR filter designed using MATLAB's `fir2` routine. A 12th-order IIR filter described in the next section and constructed in Example 4.11 is also shown for comparison.

two peaks at 75 and 225 Hz, but many other noise peaks of nearly equal amplitude are seen. In the filtered data spectrum (Figure 4.20b), the peaks are not any larger but are clearly more evident as the energy outside the two passbands has been removed. In addition, many analysis procedures would benefit from the overall reduction of noise.

4.5 Infinite Impulse Response Filters

To increase the attenuation slope of a filter, we can apply two or more filters in sequence. In the frequency domain, the frequency spectra multiply. This leads to a steeper attenuation slope in the spectrum. Figure 4.21 shows the magnitude spectrum of a 12th-order FIR rectangular window filter (dark line) and the spectra of two such filters in series (light gray line). However, for the same computational effort, you can use a single 24th-order filter and get an even steeper attenuation slope (Figure 4.21, medium gray line).

An intriguing alternative to using two filters in series is to filter the same data twice, first using a standard moving average FIR filter, then feeding back the output of the FIR filter to another filter that also contributes to the output. This becomes a moving average recursive filter with the two filters arranged as in Figure 4.22. The upper pathway is the standard FIR filter; the lower feedback pathway is another FIR filter with its own impulse response, $a[\ell]$. The problem with such an arrangement is apparent from Figure 4.22: the lower pathway receives its input from the filter's output but that input also requires the output: the output, $y[n]$, is a function of itself. To avoid the *algebraic loop* (a mathematical term for vicious circle) that such a configuration produces, the lower feedback pathway only operates on *past values* of $y[n]$; that is, values

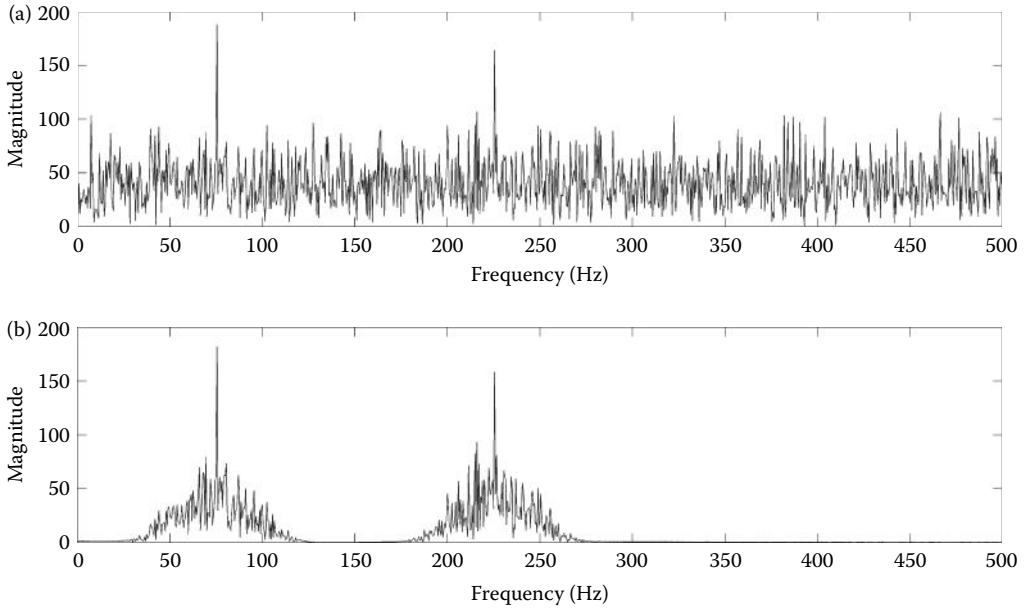

Figure 4.20 (a) Magnitude spectrum of two sinusoids buried in a large amount of noise (SNR = –20 dB). The signal peaks at 75 and 225 Hz are visible, but a number of noise peaks have the substantial amplitude. (b) The spectrum of the signal and noise after filtering with a double bandpass filter having the spectral characteristic seen in Figure 4.19b. The signal peaks are no larger than in the unfiltered data, but are much easier to see as the surrounding noise has been greatly attenuated. Other interesting filters with unusual spectra are explored in the problems.

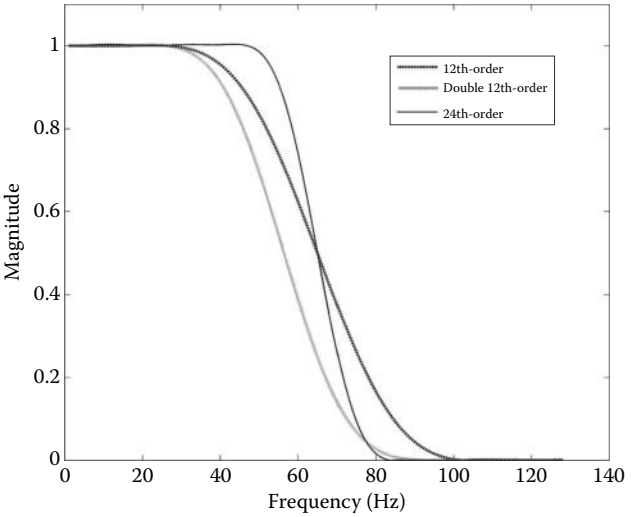

Figure 4.21 The magnitude spectrum of a 12th-order FIR filter (dark line) and the spectrum obtained by passing a signal between two such filters in series (light gray line). The two series filters produce a curve that has approximately twice the downward slope of the single filter. However, a single 24th-order filter produces an even steeper slope for the same computational effort (medium gray line).

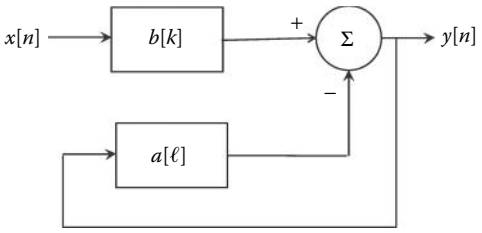

Figure 4.22 The configuration of an IIR filter containing two moving average-type FIR filters. The upper filter plays the same role as in a standard FIR filter. The lower filter operates on the output, but only past values of the output that have already been determined by the upper filter.

that have already been determined by the upper filter. (In chicken-and-egg terms, the output, or rather the *past output*, comes first.)

The equation for an IIR filter can be derived from the configuration given in Figure 4.22. The output is the convolution of the input with the upper FIR filter, minus the convolution of the lower FIR filter with the output, but only past values of the output. This tacks on an additional term to the basic FIR convolution equation given in Equation 4.14 to account for the second filter leading to Equation 4.12, and repeated here:

$$y[n] = \sum_{k=0}^{K} \underbrace{b[k]x[n-k]}_{\text{Upper path}} - \sum_{\ell=1}^{L} \underbrace{a[\ell]y[n-\lambda]}_{\text{Lower path}} \tag{4.32}$$

where $b[k]$ are the upper filter coefficients, $a[\ell]$ are the lower filter coefficients, $x[n]$ is the input, and $y[n]$ is the output. Note that while the b coefficients are summed over delays beginning at $k = 0$, the a coefficients are summed over delays beginning at $\ell = 1.0$. Hence, the a coefficients are only summed over past values of $y[n]$. Since part of an IIR filter operates on passed values of the output, these filters are recursive filters, and the a coefficients are the *recursive coefficients*. The primary advantages and disadvantages of FIR and IIR filters are summarized in Table 4.1.

4.5.1 IIR Filter Implementation

IIR filters are applied to signals using Equation 4.32, where x is the signal and a and b are the coefficients that define the filter. While it would not be difficult to write a routine to implement this equation, it is again unnecessary as the MATLAB filter routine works for IIR filters as well:

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

where `b` and `a` are the same filter coefficients used in Equation 4.32, `x` is the input signal, and `y` is the output.

4.1 FIR versus IIR Filters: Features and Applications

Filter Type	Features	Applications
FIR	Easy to design Stable Applicable to 2-D data (i.e., images)	Fixed, 1-D 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, 1-D filters, particularly at low cutoff frequencies Real-time applications where speed is important

As mentioned above, the time shift of either FIR or IIR filters can be eliminated by using a noncausal implementation. Eliminating the time shift also reduces the filter's phase shift, and noncausal techniques can be used to produce zero-phase filters. Since noncausal filters use both *future* as well as past data samples (future only with respect to a given sample in the computer), they require the data to exist already in computer memory. The Signal Processing Toolbox routine `filtfilt` employs noncausal methods to implement filters with no phase shift. The calling structure is exactly the same as `filter`:

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

Several problems dramatically illustrate the difference between the use of `filter` and `filtfilt`.

4.5.2 Designing IIR Filters with MATLAB

The design of IIR filters is not as straightforward as that for FIR filters. However, the MATLAB Signal Processing Toolbox provides a number of advanced routines to assist in this process. IIR filters are similar to 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 adds 20 dB/decade to the slope. In IIR digital filters, the first a coefficient, $a[0]$, always equals 1.0, but each additional a coefficient adds 20 dB/decade to the slope. So an eighth-order analog filter and an eighth-order IIR filter have the same slope. Since the downward slope of an IIR filter increases by 20 dB/decade for each increase in filter order, determining the filter order needed for a given attenuation is straightforward.

IIR filter design with 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. As with FIR design, a single-stage design process can be used if the filter order is known, and that is the approach that is covered here.

The Yule–Walker recursive filter is the IIR equivalent of the `fir2` FIR filter routine in that it allows for the specification of a general desired frequency response curve. The calling structure is also very similar to that of `fir2`.

```
[b,a] = yulewalk(order,f,G);  % Find coeff. of an IIR filter
```

where `order` is the filter order, and `f` and `G` specify the desired frequency characteristic in the same manner as `fir2`: `G` is a vector of the desired filter gains at the frequencies specified in `f`. As in all MATLAB filter design routines, the frequencies in `f` are relative to $f_s/2$, and the first point in `f` must be 0 and the last point 1. Again, duplicate frequency points are allowed, corresponding to steps in the frequency response.

EXAMPLE 4.11

Design the double bandpass filter that is used in Example 4.10. The first passband has a range of 50–100 Hz and the second passband ranges between 200 and 250 Hz. Use an IIR filter order of 12 and compare the results with the 65th-order FIR filter used in Example 4.10. Plot the frequency spectra of both filters, superimposed for easy comparison.

Solution

Modify Example 4.10 to add the Yule–Walker filter, determine its spectrum using `freqz`, and plot it superimposed with the FIR filter spectrum. Remove the code relating to the signal as it is not needed in this example.

```
% Example 4.11 Design a double bandpass IIR filter and compare with
% a similar FIR filter
%
       ..... same initial code as in Example 4.10......
% Design filter
f = [0 50 50 100 100 200 200 250 250 fs/2]/(fs/2); % Construct desired
G = [0 0 1 1 0 0 1 1 0 0];                          % frequency curve
b1 = fir2(L1,f,G);                        % Construct FIR filter
[H1,f1] = freqz(b1,1,512,fs);             % Calculate FIR filter spectrum
[b2 a2] = yulewalk(L2,f,G);               % Construct IIR filter
[H2 f2] = freqz(b2,a2,512,fs);            % Calculate IIR filter spectrum
%
plot(f1,abs(H1),'k'); hold on;            % Plot FIR filter mag. spectrum
plot(f2,abs(H2),':k','LineWidth',2);      % Plot IIR filter magnitude spectrum
 .......labels........
```

The spectral results from two filters are shown in Figure 4.19b. The magnitude spectra of both filters look quite similar despite that fact that the IIR filter has far fewer coefficients. The FIR filter has 65 *b* coefficients and 1 *a* coefficient while the IIR filter has 13 *b* coefficients and 13 *a* coefficients.

In an extension of Example 4.11, 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. Surprisingly, despite the larger number of coefficients, the FIR filter is faster than the IIR filter: 0.077 ms for the FIR versus 1.12 ms for the IIR filter. However, if `conv` is used to implement the FIR filter, the FIR filter takes longer: 2.64 ms.

Several well-known analog filter types can be duplicated as IIR filters. Specifically, analog filters termed *Butterworth*, *Chebyshev* type I and II, and *Elliptic* (or Cauer) designs can be implemented as IIR digital filters and are supported in 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, as shown in Figure 1.12, the Butterworth filter has a less sharp initial attenuation characteristic than the Chebyshev filter. The Chebyshev type I filter shown in Figure 1.12 features a faster rolloff than the Butterworth filter, but has ripple in the passband. The Chebyshev type II filter has ripple only in the stopband, its passband is monotonic, but it does not rolloff as sharply as type I. 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, but have equiripple in both the passband and stopband. While 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, making Butterworth the filter of choice.

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 $f_s/2$.) If the `'ftype'` argument is missing, a lowpass filter is designed provided `wn` is scalar; if `wn` is a two-element vector, then a bandpass filter is designed. In the latter case, `wn = [w1 w2]`, where `w1` is the low cutoff frequency and `w2` is the high cutoff frequency. If a highpass filter is desired, then `wn` should be a scalar and `'ftype'` should be `'high'`. For a stopband filter, `wn` should be 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.

While the Butterworth filter is the only IIR filer 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 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(order,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 does not exceed 40 dB *below* the passband gain. In effect, this value specifies the minimum attenuation in the stopband. The elliptic filter includes both stopband and passband ripple values:

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

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

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

EXAMPLE 4.12

Plot the frequency response curves (in dB versus log frequency) obtained from an eighth-order lowpass filter using the Butterworth, Chebyshev type 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 design routines above to determine the *a* and *b* coefficients, use `freqz` to calculate the complex frequency spectrum, take the absolute value of this spectrum and convert to dB `20*log10(abs(H))`. Plot using `semilogx` to put the horizontal axis in term of log frequency.* Repeat this procedure for the four filters.

```
% Example 4.12 Frequency response of four IIR 8th-order lowpass filters
%
N = 256;                % Padding
fs = 2000;              % Sampling frequency
L = 8;                  % Filter order
fc = 200/(fs/2);        % Filter cutoff frequency
rp = 3;                 % Maximum passband ripple in dB
rs = 60;                % Stopband ripple in dB
%
```

* Most spectral plots thus far have been in linear units. Plots of dB versus log frequency are often used for filter spectra, particularly IIR filter spectra, as the attenuation slopes are in multiples of 20 dB/decade. These slopes become straight lines in dB versus log *f* plots. See Figure 1.10 for a comparison of linear and dB versus log *f* (i.e., log–log) plots. We use both plot types in this book.

```
% Determine filter coefficients
[b,a] = butter(L,fc);
[H,f] = freqz(b,a,N,fs);
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.......
%
[b,a] = cheby1(L,rp,fc);    % Chebyshev Type I filter coefficients
[H,f] = freqz(b,a,N,fs);    % Calculate complex spectrum
H = 20*log10(abs(H));       % Convert to magnitude in dB
 .........plot as above, labels and title.......
%
[b,a] = cheby2(L,rs,fc);    % Chebyshev Type II filter coefficients
.......Use freqz, dB conversion and plot as above, labels and title....
%
[b,a] = ellip(L,rp,rs,fc);  % Elliptic filter coefficients
.......Use freqz, dB conversion and plot as above, labels and title....
```

The spectra of the four filters are shown in Figure 4.23. As described above, 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

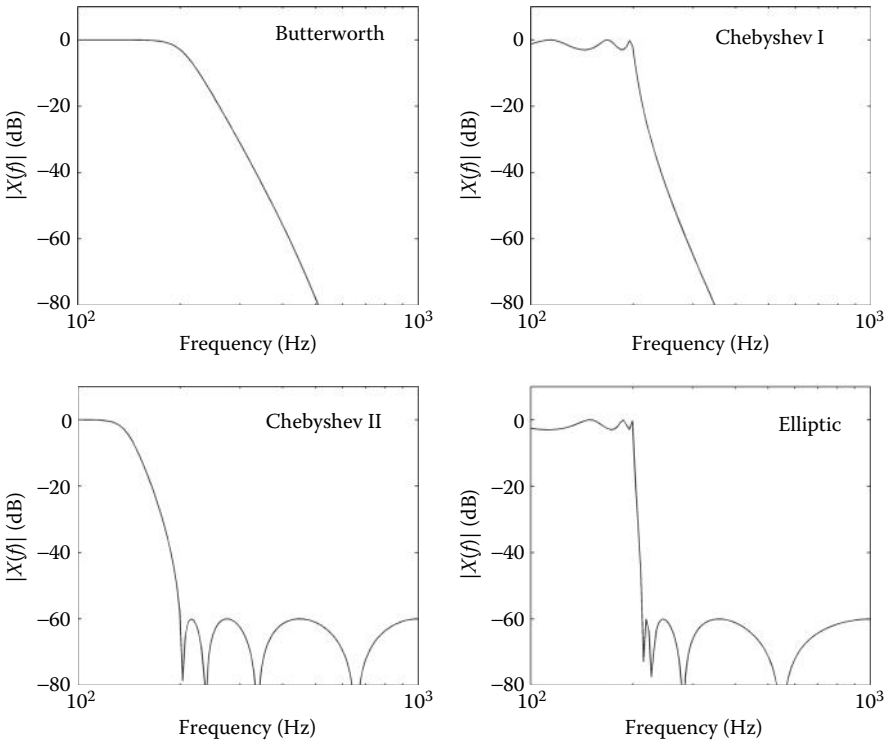

Figure 4.23 The spectral characteristics of four different eighth-order IIR lowpass filters with a cutoff frequency of 200 Hz. The assumed sampling frequency is 2 kHz. The spectral characteristics show that it is possible to increase the initial sharpness of an IIR filter significantly if various types of ripple in the spectral curve can be tolerated.

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 ripple is found in both the passband and stopband. Reducing the ripple of these last three filters is possible in the design process, but this also reduces the filter's initial sharpness, another illustration of the compromises that continually arise in engineering.

Other types of filters exist in both FIR and IIR forms; however, the filters described above are the most common and are the most useful in biomedical engineering.

4.6 Summary

Most linear approaches to noise reduction are based on averaging. In ensemble averaging, entire signals are averaged together. This approach is quite effective, but requires multiple versions, or observations, of the same signal and a timing reference for the signals. If repetitive stimuli produce similar responses, time-locked to the stimulus, they are ideal for ensemble averaging. The most noted use of averaging in biosignal analysis is to separate out neuroelectric-evoked responses from the background noise in the EEG signal.

A special digital transfer function based on the Z-transform can be used to analyze digital processes such as IIR filters. The spectral characteristics of Z-domain transfer functions can easily be determined using the Fourier transform. While it is not easy to convert between the Z-domain transfer function and the Laplace transfer function, it is possible to convert to the frequency domain by substituting $e^{j\omega}$ for z.

Filters are used to shape the spectrum of a signal, often to eliminate frequency ranges that include noise or to enhance frequencies of interest. Digital filters are also based on averaging applied as a moving average and/or as a recursive moving average. FIR filters are straightforward moving average processes, usually with unequal filter weights. The filter weights correspond to the impulse response of the filter, so they can be implemented by convolving the weights with the input signal. FIR filters have linear phase characteristics and they can be applied to 2-D data such as images. FIR filters can be designed based on the inverse Fourier transform, as in rectangular window filters, but MATLAB routines also exist that simplify design.

IIR filters combine a moving average process with a recursive moving average to achieve much sharper attenuation characteristics for the same number of filter coefficients. In IIR filters, the moving average impulse response is applied to the input signal through standard convolution, while a recursive moving average is applied to a delayed version of the output also using convolution. IIR filters can be designed to mimic the behavior of analog filters; some types can produce sharp initial cutoffs at the expense of some ripple in the bandpass region. An IIR filter known as the Butterworth filter produces the sharpest initial cutoff without bandpass 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 support routines.

PROBLEMS

4.1 Apply ensemble averaging to the data `ensemble_x.mat`. This file contains a data matrix labeled `x`. The data matrix contains 100 responses of an exponential signal buried in noise. In this matrix, each row is a separate response, so no transposition is necessary. Plot two randomly selected samples of these responses. Is it possible to analyze the buried signal in any single record? Construct and plot the ensemble average for this data. Scale the time axis correctly assuming $T_s = 0.05$.

4.2 Expand Example 4.1 to evaluate ensemble averaging for different numbers of individual responses. Load file `ver_problem.mat`, which contains the VER data, `ver`,

along with the actual, noise-free evoked response in `actual_ver`. The visual response data set consists of 1000 responses, each 500 points long. The sample interval is 10 ms. Construct an ensemble average of 25, 100, and 1000 responses. The variables are in the correct orientation and do not have to be transposed. Subtract the noise-free variable (`actual_ver`) from an individual-evoked response and the three ensemble averages to get an estimate of the noise in the three waveforms. Compare the standard deviations of one of the unaveraged with the three averaged waveforms. Output the theoretical standard deviation based on Equation 4.3 using the unaveraged standard deviation and the appropriate number of averages. Compare the theoretical and actual noise values. Also plot one of the individual responses and, separately, the signal produced by averaging 1000 responses. Superimpose the noise-free response on the averaged signal and compare.

4.3 This program illustrates one of the problems that can occur with ensemble averaging: the lack of a fixed and stable reference signal. The file `ver_problem2.mat` contains three variables: `actual_ver`, which is the noise-free VER, `ver`, which consists of 100 noise records recorded with a fixed reference signal, and `ver1`, which consists of 100 records recorded with a reference signal that varies randomly by ±150 ms. Construct ensemble averages from `ver` and `ver1` and plot separately along with the noise-free record. $T_s = 0.005$ s. What is the difference in the two averages?

4.4 Find the spectrum (magnitude and phase) of the system represented by the Z-transform:

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

Use Equation 4.13 to find the spectrum (not `freqz`). Plot magnitude and phase (in deg) versus frequency (in Hz) assuming $f_s = 500$ Hz. Also find the step response of this system over a time period of 0.5 s.

4.5 Write the Z-transform equation for a fourth-order Butterworth highpass filter with a relative cutoff frequency of 0.3. [Hint: Get the coefficients from MATLAB's `butter` routine.]

4.6 Find the spectrum (magnitude and phase) of the system represented by the Z-transform:

$$H(z) = \frac{0.42x10^{-3} + 1.25x10^{-3}z^{-1} + 1.25x10^{-3}z^{-2} - 0.42x10^{-3}z^{-3}}{1 - 2.686z^{-1} + 2.42z^{-2} - 0.73z^{-3}}$$

Plot the magnitude in dB versus log f and the phase (in deg) versus log f assuming $f_s = 2000$ Hz. Limit the magnitude plot to be between 0 and −80 dB. Also find the step response of this system over a time period of 0.1 s. Use Equation 4.13 to find the spectrum (not `freqz`). Assuming $H(z)$ is a filter, what can you say about the filter from the magnitude plot?

4.7 Use MATLAB to find the frequency response of a 3-point moving average filter (i.e., $b = [1\ 1\ 1]/3$) and a 10-point moving average filter. Use appropriate zero-padding to improve the spectra. Plot both the magnitude and phase spectra assuming a sample frequency of 500 Hz.

4.8 Use `sig_noise` to generate a 20-Hz sine wave in 5 dB of noise (i.e., SNR = −5 dB) and apply the two filters from Problem 4.7 using the MATLAB `filter` routine. Plot

the time characteristics of the two outputs. Use a data length (N) of 200 in `sig_noise` and remember that `sig_noise` assumes a sample frequency of 1 kHz.

4.9 Find the magnitude spectrum of an FIR filter with a weighting function of b = [.2 .2 .2 .2 .2] in two ways: (a) apply the `fft` with padding to the filter coefficients as in Problem 4.7 and plot the magnitude spectrum of the result; (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; that is, `y = conv(b, randn(20000,1))`. Use the Welch averaging method described in Section 4.3 to smooth the spectrum. For the Welch method, use a segment size of 128 points and a 50% segment overlap. Since the `pwelch` routine produces the power spectrum, you need to take the square root to get the magnitude spectrum for comparison with the FT method. The two methods use different scaling, so the vertical axes are slightly different. Assume a sampling frequency of 200 Hz for plotting the spectra.

4.10 Use `sig_noise` to construct a 512-point array consisting of two closely spaced sinusoids of 200 and 230 Hz with SNR of −8 dB and −12 dB, respectively. Plot the magnitude spectrum using the FFT. Generate a 25 coefficient rectangular window bandpass filter using approach in Example 8.5. Set the low cutoff frequency to 180 Hz and the high cutoff frequency to 250 Hz. Apply a Blackman–Harris window to the filter coefficients, then filter the data using MATLAB's filter routine. Plot the magnitude spectra before and after filtering.

4.11 Use `sig_noise` to construct a 512-point array consisting of a single sinusoid at 200 Hz in −20 dB noise (a very large amount of noise). Narrowband filter the data around 200 Hz with a rectangular window filter. Use high and low cutoff frequencies at 180 and 220 Hz and use 25 coefficients in the filter. Plot the power spectrum obtained using the FFT direct method before and after filtering and using the Welch method. For the Welch method, use a segment length of 128 samples with 50% overlap. Note the improvement due to filtering in the ability to identify the narrowband component, particularly with the Welch method.

4.12 This problem compares the Blackman–Harris and Hamming windows in the frequency domain. Write a program to construct the coefficients of a lowpass rectangular window filter with a cutoff frequency of 200 Hz. Make the filter length L = 33 coefficients. Assume f_s = 1000 Hz. Generate and apply the appropriate length Hamming (Equation 4.24) and Blackman–Harris (Equation 4.25) windows to the filter coefficients, $b[k]$. Construct the two windows from the basic equations as in Example 4.5. Find and plot the spectra of the filter without a window and with the two windows. Plot the spectra superimposed to aid comparison and pad to 256 samples. As always, do not plot redundant points. Note that the Hamming window produces a somewhat steeper attenuation, but has just the slightest ripple before the cutoff.

4.13 This problem compares the Blackman–Harris and Hamming windows in the time domain. Construct the rectangular window filter used in Problem 4.12, but make L = 129 coefficients and the cutoff frequency 100 Hz. As in Problem 4.12, generate and apply the appropriate length Hamming (Equation 4.24) and Blackman–Harris (Equation 4.25) windows to the filter coefficients, $b[k]$. Determine the impulse response of these two filters using the MATLAB filter routine with an impulse input. The impulse input should consist of a 1 followed by 255 zeros. Plot the impulse responses superimposed (in different colors) and you will find they look nearly identical. Limit the time axis to 0.1 s to better observe the responses. Subtract the impulse

response of the Blackman–Harris-windowed filter from the impulse response of the Hamming-windowed filter and plot the difference. Note that, while there is a difference, it is several orders of magnitude less than the impulse responses.

4.14 Load file `ECG _ 9.mat`, which contains 9 s 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 due to respiration artifact. Filter the data with a highpass filter having 65 coefficients and a cutoff frequency of 8 Hz. Use the window of your choice to taper the filter weights. Plot the magnitude spectrum of the filter to confirm the correct type and cutoff frequency. Also plot the filtered and unfiltered ECG data. [Hint: You can modify a section of the code in Example 4.4 to generate the highpass filter.]

4.15 ECG data are often used to determine the heart rate by measuring the time interval between the peaks that occur in each cycle known as the *R wave*. To determine the position of this peak accurately, ECG data are first prefiltered with a bandpass filter that enhances the *R* wave and the two small negative peaks on either side, the *Q* and *S* wave. The three waves together are termed the *QRS complex*. Load file `ECG_noise.mat`, which contains 10 s of noisy ECG data in variable `ecg`. Filter the data with a 65-coefficient FIR bandpass filter to best enhance the *R*-wave peaks. Use the tapering window of your choice. (Note that it is not possible to eliminate all the noise, just improve the identification of the peaks.) Determine the low and high cutoff frequencies empirically, but they will be in the range of 4–24 Hz. The sampling frequency is 250 Hz.

4.16 This problem explores the use of multiple filters in sequence. Load the file `deriv1_data.mat` containing the signal variable `x`. Take the derivative of this signal using the two-point central difference algorithm with a skip factor of 6, implemented using the `filter` routine. Now add an additional lowpass filter using a rectangular window filter with 65 coefficients and a cutoff frequency 25 Hz. Use the `conv` routine with option `'same'` to eliminate the added delay that is induced by the filter routine. Plot the original time data, the result of the two-point central difference algorithm, and the lowpass filtered derivative data. Note the clear derivative trace in the lowpass filtered data. Also plot the spectrum of the two-point central difference algorithm, the lowpass filter, and the combined spectrum. The combined spectrum can be obtained by simply multiplying the two-point central difference spectrum point-by-point with the lowpass filter spectrum.

4.17 Load the file `deriv1_data.mat`, which contains signal variable `x` ($f_s = 250$ Hz). Use these data to compare the two-point central difference algorithm with the combination of a difference (Equation 4.29) and a lowpass filter. Use a skip factor of 8 for the two-point central difference algorithm and with the difference operator use a 67th-order rectangular window lowpass filter with a cutoff frequency of 20 Hz. Use MATLAB's `diff` to produce the difference output, and scale the result by dividing by T_s. Filter this output with the lowpass filter. Note that the derivative obtained this way is smooth, but has low-frequency noise. Again, this problem shows the application of two sequential operations.

4.18 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. Use MATLAB's `fir2` to design a 65-coefficient FIR filter having a spectrum with a double bandpass. The bandwidth of the two bandpass regions should be 20 Hz centered about the two peaks. Plot the filter's magnitude spectrum superimposed on the desired spectrum. [Recall: Referring to the calling structure of `fir2`, plot `G` versus `f` after scaling the frequency vector, `f`, appropriately.] Also plot the signal's magnitude spectrum before and after filtering.

4.19 The file `ECG_60HZ_data.mat` contains an ECG signal in variable `x` that was sampled at $f_s = 250$ Hz and has been corrupted by 60-Hz noise. The 60-Hz noise is at a high frequency compared to the ECG signal, so it appears as a thick line superimposed on the signal. Construct a 127-coefficient FIR rectangular *bandstop* filter with a center frequency of 60 Hz and a bandwidth of 10 Hz (i.e., ±5 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 also plot the filter's magnitude spectrum to ensure the filter spectrum is what you want. [Hint: You can easily modify a portion of the code in Example 4.7 to implement the bandstop filter.]

4.20 This problem compares causal and noncausal FIR filter implementation. Generate the filter coefficients of a 65th-order rectangular window filter with a cutoff frequency of 40 Hz. Apply a Blackman–Harris window to the filter. Then apply the filter to the noisy 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 `filter` routine and noncausal `conv` with the `'same'` option. (Without this option, `conv` is like `filter` except that it produces extra points.) Plot the two waveforms along with the original, superimposed for comparison. Note the obvious differences. Also note that while the filter removes much of the noise, it also reduces the sharpness of the transitions.

4.21 Given the advantage of a noncausal filter with regard to the time shift shown in Problem 4.20, why not use noncausal filters routinely? This problem shows the downsides of noncausal FIR filtering. Generate the filter coefficients of a 33rd-order rectangular window filter with a cutoff frequency of 100 Hz, assuming $f_s = 1$ kHz. Use a Blackman–Harris window on the truncated filter coefficients. Generate an impulse function consisting of a 1 followed by 255 zeros. Now apply the filter to the impulse function in two ways: causally using the MATLAB `filter` routine, and noncausally using the `conv` routine with the `'same'` option. (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. (For a change, you can plot the phase in radians.) 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 noncausal filter has truncated 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, although the phase curves are different due to the different delays produced by the two filters. The phase spectrum of the noncausal filter shows reduced phase shift with frequency as would be expected. This problem demonstrates that noncausal filters can create artifact with the initial portion of an input signal because of the way it compensates for the time shift of causal filters.

4.22 Compare the step response of an 8th-order Butterworth filter and a 44th-order (i.e., $L = 44 + 1$) rectangular window (i.e., FIR) filter, both having a cutoff frequency of $0.2\,f_s$. Assume a sampling frequency of 2000 Hz for plotting. (Recall that MATLAB routines normalize frequencies to $f_s/2$, while the rectangular window filters equations are normalized to f_s). Use a Blackman–Harris window with the FIR filter, then implement both filters using MATLAB's `filter` routine. Use a step of 256 samples but offset 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 magnitude spectra of both filters. Use Equation 4.13 to find the spectrum of both filters. (For the FIR filter, the denominator of Equation 4.13 is 1.0.) Note that although the magnitude

spectra have about the same slope, the step response of the IIR filter has more overshoot—another possible downside to IIR filters.

4.23 Repeat Problem 4.19, but use an eighth-order Butterworth bandstop filter to remove the 60-Hz noise. Load the file `ECG_60HZ_data.mat` containing the noisy ECG signal in variable `x` (f_s = 250 Hz). Apply the filter to the noisy signal 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. [Recall that with the Signal Processing Toolbox, cutoff frequencies are specified relative to $f_s/2$.] Note how effective the low-order IIR filter is at removing the 60-Hz noise.

4.24 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 that 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. Also plot the noise-free signal found as `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.

4.25 This problem is similar to Problem 4.21 in that it illustrates problems with noncausal filtering, except that an IIR filter is used and the routine `filtfilt` is used to implement the noncausal 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 this, 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. However, even after changing the delay, the noncausal implementation has a small amount of ripple in the passband of the magnitude spectrum.

4.26 Load the data file `ensemble_data.mat`. Filter the average with a 12th-order Butterworth filter. (The sampling frequency is not given, but you do not need it to work the problem.) 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, where the interesting part is not near the edges, the noncausal filter is appropriate. Use MATLAB's `title` or `text` command to display the cutoff frequency on the plot containing the filtered data.

4.27 This problem compares FIR–IIR filters. Construct a 12th-order Butterworth highpass filter with a cutoff frequency of 80 Hz assuming f_s = 300 Hz. Construct an FIR highpass filter having the same cutoff frequency using Equation 4.26. Apply a Blackman–Harris window to the FIR filter. Plot the spectra of both filters and adjust the order of

the FIR filter to approximately match the slope of the IIR filter. Use Equation 4.13 to find the spectra of both filters. Count and display the number of b coefficients in the FIR filter and the number of a and b coefficients in the IIR filter.

4.28 Find the power spectrum of an LTI system four ways: (1) use white noise as the input and take the Welch power spectrum of the output; (2) use white noise as an input and take the Fourier transform of the autocorrelation function of the output; (3) use white noise as an input and take the Welch power spectrum of the cross-correlation of the output with the input; and (4) apply Equation 4.13 to the a and b coefficients. The third approach works even if the input is not white noise. As a sample LTI system, use a fourth-order Butterworth bandpass filter with cutoff frequencies of 150 and 300 Hz. For the first three methods, use a random input array of 20,000 points. Use `xcorr` to calculate the auto- and cross-correlation and `pwelch` to calculate the power spectrum. For `pwelch`, use a window of 128 points and a 50% overlap. [Owing to the number of samples, this program may take 30 s or more to run so you may want to debug it with fewer samples initially.] Note that while the shapes are similar, the vertical scales will be different.

4.29 Load the file `nb2_noise_data.mat`, which contains signal variable `x`. This signal has two sinusoids at 200 and 400 Hz in a large amount of noise. Narrowband filter the signal around 200 and 400 Hz using a double bandpass IIR Yule–Walker filter. Use a 12th-order filter with high and low cutoff frequencies at ±30 Hz of the center frequencies. Plot the power spectrum obtained using the Welch method before and after filtering. Use a segment length of 256 samples with 50% overlap. Note the substantial improvement in the ability to identify the narrowband signals.

4.30 Find the step responses of the four IIR filters presented in Example 4.12. Use the same filter parameters used in that example and implement using `filter`. Use a step signal of 90 samples. For plotting the step responses, use $f_s = 2$ kHz as in Example 4.12. Note the unusual step responses produced by the Chebyshev type 1 and elliptic filters. Also note that even the Butterworth produces some overshot in the step response.