3

Spectral Analysis
Classical Methods

3.1 Introduction

Sometimes, the frequency content of the waveform provides more useful information than the time-domain representation. Many biological signals demonstrate interesting or diagnostically useful properties when viewed in the so-called *frequency domain*. Examples of such signals include heart rate, EMG, EEG, ECG, eye movements and other motor responses, acoustic heart sounds, stomach, and intestinal sounds. In fact, just about all biosignals have, at one time or another, been examined in the frequency domain. Figure 3.1 shows the time response of an EEG signal and an estimate of spectral content using the classical Fourier transform (FT) method described later. Several peaks in the frequency plot can be seen, indicating significant energy in the EEG at these frequencies. Most of these peaks are associated with general neurological states.

Determining the frequency content of a waveform is termed *spectral analysis* and the development of useful approaches for this frequency decomposition has a long and rich history. Spectral analysis decomposes a time-domain waveform into its constituent frequencies just as a prism decomposes light into its spectrum of constituent colors (i.e., specific frequencies of the electromagnetic spectrum).

Various techniques exist to convert time-domain signals into their spectral equivalent. Each has different strengths and weaknesses. Basically, spectral analysis methods can be divided into two broad categories: classical methods based on the Fourier transform (FT), and modern methods such as those based on models of the signal's source. The accurate determination of the waveform's spectrum requires that the signal must be periodic, of finite length, and noise free. Unfortunately, many biosignals are sufficiently long that only a portion is available for analysis. Moreover, biosignals are often corrupted by substantial amounts of noise or artifact (see Section 2.2). If only a portion of the actual signal can be analyzed and/or if the waveform contains noise along with the signal, then all spectral analysis techniques must necessarily be approximate, that is, they are *estimates* of the true spectrum. The modern spectral analyses described in Chapter 5 attempt to improve the estimation accuracy of specific spectral features, but require some knowledge, or guesswork, about the signal content.

An intelligent application of spectral analysis techniques requires an understanding of what spectral features are likely to be of interest and which methods provide the most accurate determination of these features. Two spectral features of potential interest are the overall shape of

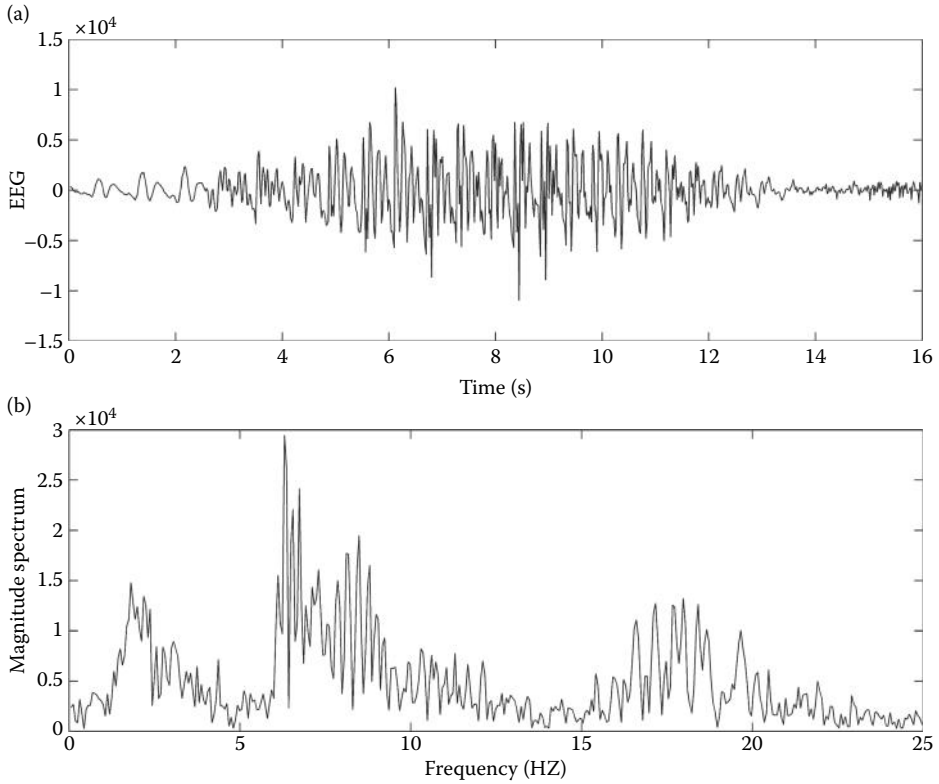

Figure 3.1 (a) Segment of an EEG time-domain signal also shown in Figure 2.16. (b) The equivalent magnitude component of the frequency-domain representation. The peaks observed in this spectrum are known to correspond with certain neurological states. This spectrum is calculated in Example 3.1.

the spectrum, termed the spectral estimate, and/or local features of the spectrum, sometimes referred to as parametric estimates. Local features would include narrowband signals whereas spectral estimates would describe the broadband characteristics of a signal. The techniques that provide good spectral estimation are poor local estimators and vice versa. Fortunately, we are not usually interested in the accurate estimation of both narrowband and broadband features, but if we are, then different analyses will be required.

Figure 3.2a shows the spectral estimate obtained by applying the traditional FT to a waveform consisting of a 100-Hz sine wave buried in white noise. The SNR is –14 dB, that is, the signal amplitude is only 1/5 of the noise. Note that the 100-Hz sine wave is readily identified as a peak in the spectrum at that frequency, but many other smaller peaks are seen that are due to noise. Figure 3.2b shows the spectral estimate obtained by a smoothing process applied to the same signal. This smoothing operation is called the Welch method and is described later in this chapter. The resulting spectrum provides a more accurate representation of the overall spectral features (predominantly those of the white noise), but the 100-Hz signal is lost. Figure 3.2 shows that the smoothing approach is a good spectral estimator in the sense that it provides a better estimate of the dominant component (i.e., noise), but it is not a good signal detector of local features.

The classical procedures for spectral estimation are described in this chapter with particular regard to their strengths and weaknesses. These methods are easily implemented in MATLAB. Modern methods for spectral estimation are covered in Chapter 5.

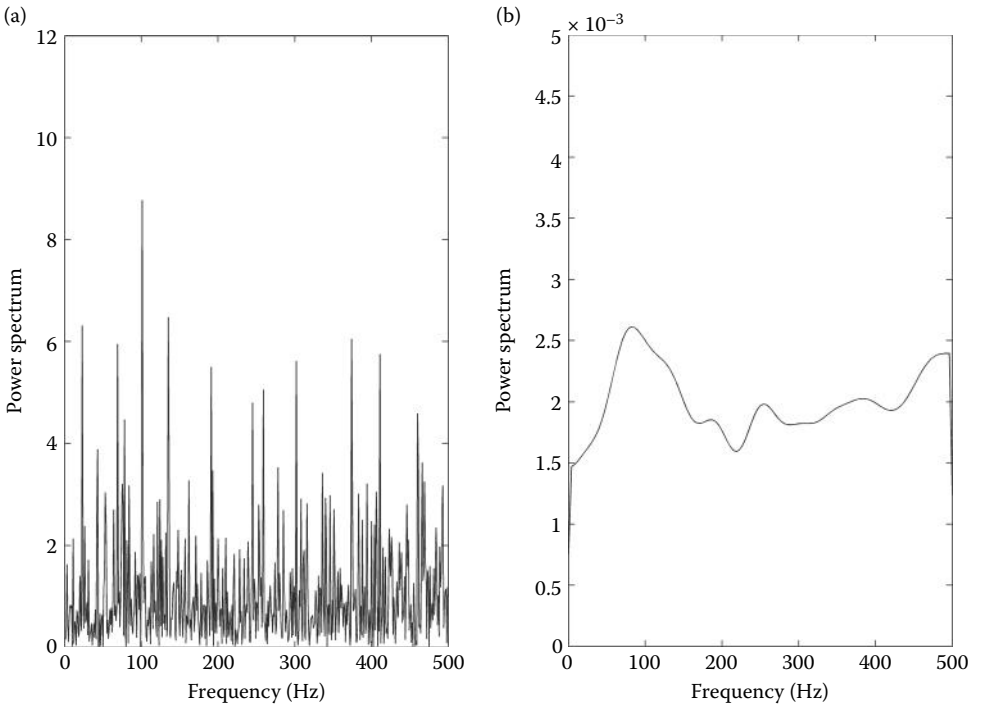

Figure 3.2 Spectra obtained from a waveform consisting of a 100-Hz sine wave and white noise using two different methods. (a) The spectrum produced by the FT clearly shows the signal as a spike at 100 Hz. However, other spikes are present that are just noise. (b) An averaging technique creates a smooth spectrum that represents the background white noise (that should be flat) better, but the 100-Hz component is no longer clearly visible.

3.2 Fourier Series Analysis

Of the many techniques used for spectral estimation, the classical method commonly known as the Fourier transform* is the most straightforward. Essentially, FT approaches use the sinusoid as a gateway between the time and frequency domains. Since sinusoids contain energy at only one frequency, a sinusoid maps to the frequency domain in a very straightforward manner. As shown in Figure 3.3, the amplitude of the sinusoid maps to a single point on the spectrum's magnitude curve and the phase of the sinusoid maps to a single point on the spectrum's phase curve. Although up to now, we have dealt with only the magnitude spectrum, a full spectrum consists of both a magnitude and phase component. Often, only the magnitude spectrum is of interest, but sometimes, we would like to know the phase characteristics. Moreover, the full spectrum is essential if we want to convert back from the frequency domain to the time domain.

If a waveform can be decomposed into sinusoids of different frequencies, this simple mapping procedure can be used to determine the magnitude and phase spectrum of the waveform. This strategy is illustrated in Figure 3.4 for a triangular waveform (left panel) that is decomposed into three sinusoids of increasing frequency (middle panel) that are then mapped to a magnitude and phase spectrum (right panels). In essence, sinusoids form a bridge between the time- and frequency-domain representations of a signal.

* The term "Fourier transform" is a general term that is loosely used to cover several related analysis techniques. These different techniques are summarized in Tables 3.2 and 3.3 along with the proper terminology.

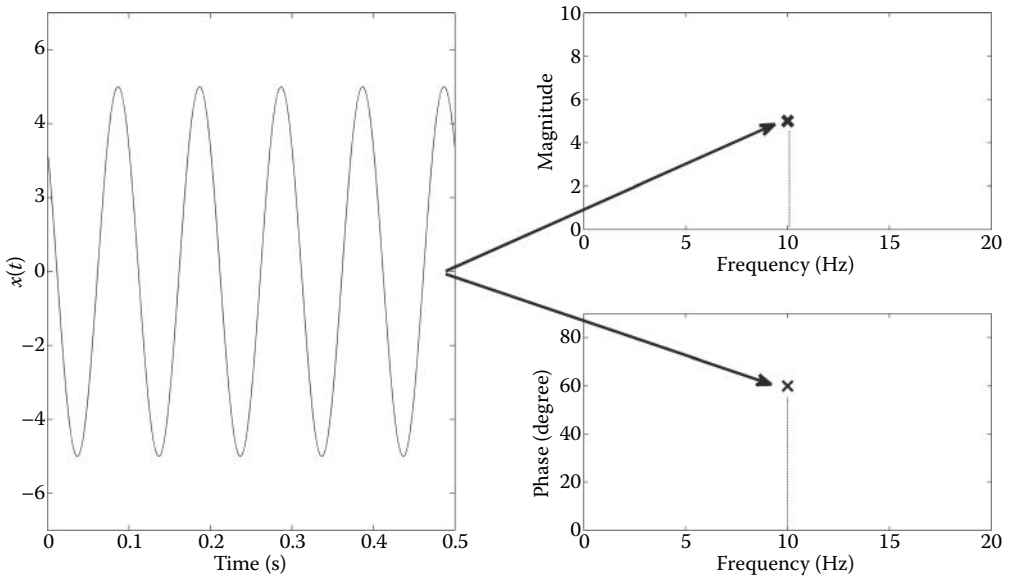

Figure 3.3 Since sinusoids contain energy at only a single frequency, they map to the frequency domain in a very simple manner. The amplitude of the sinusoid maps to a single point on the magnitude spectrum curve and the phase angle of the sinusoid maps to a single point on the phase portion of the spectrum. Both points are at the frequency of the sinusoid, in this case 10 Hz.

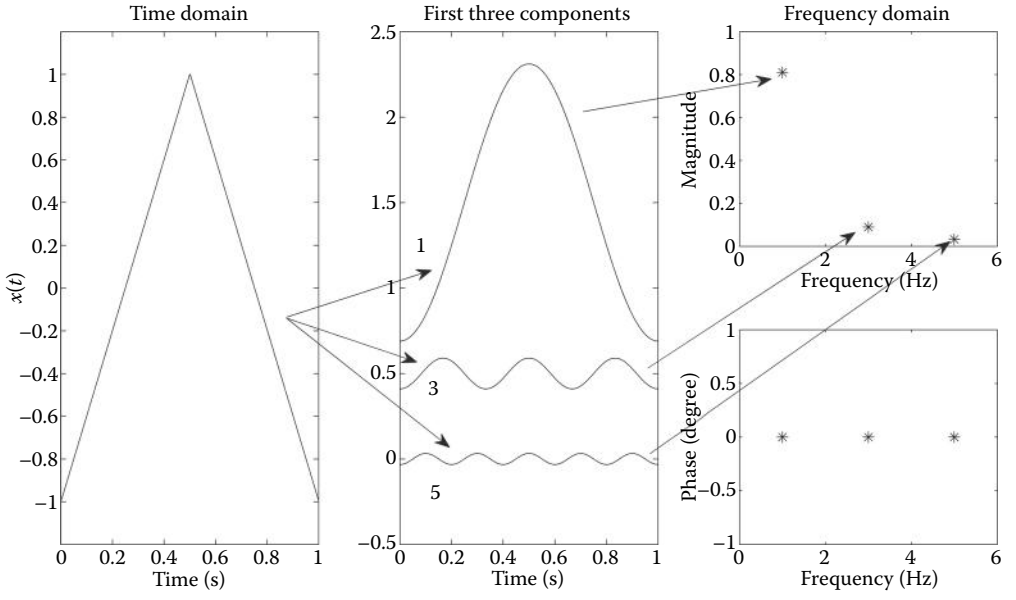

Figure 3.4 Sinusoids can be used as a gateway between a general waveform (left panel) and its frequency representation or spectrum (right panels). The triangular waveform is decomposed into three sinusoids (middle panel). These are then mapped to the magnitude spectrum (arrows) and the phase spectrum.

3.2.1 Periodic Functions

When the waveform of interest is periodic, an approach known as *Fourier series analysis* is used to convert between the time and frequency domains. It uses the "sinusoidal bridge" approach to decompose a general periodic waveform into sinusoids. The foundation of this approach is the *Fourier series theorem*, which states that any periodic waveform, no matter how complicated, can be decomposed into sinusoids that are at the same frequency as, or multiples of, the waveform's frequency. This family or basis can be expressed as either a series of sinusoids of appropriate amplitude and phase angle or a mathematically equivalent series of sines and cosines. In either case, it is only necessary to project (i.e., correlate) the waveform with the basis (see Equation 2.37). Substituting a basis of sines and cosines for $f_m(t)$ in the continuous version of the correlation equation, Equation 2.37 leads to

$$a[m] = \frac{2}{T}\int_0^T x[t]\cos(2\pi m f_1 t)dt \; m = 1,2,3,\ldots \tag{3.1}*$$

$$b[m] = \frac{2}{T}\int_0^T x[t]\sin(2\pi m f_1 t)dt \; m = 1,2,3,\ldots \tag{3.2}$$

where $f_1 = 1/T$, T is the period of the waveform $x(t)$, and m is a set of integers: $m = 1, 2, 3, \ldots$ (possibly infinite) defining the family member. This gives rise to a basis of sines and cosines having harmonically related frequencies, mf_1. If this concept was implemented on a computer, the discrete version of these equations would be used where integration becomes summation, $x(t) \rightarrow x[n]$, and $t \rightarrow nT_s$:

$$a[m] = \sum_{n=1}^{N} x[n]\cos(2\pi m f_1 nT_s) \quad m = 1,2,3,\ldots M \tag{3.3}$$

$$b[m] = \sum_{n=1}^{N} x[n]\sin(2\pi m f_1 nT_s) \quad m = 1,2,3,\ldots M \tag{3.4}$$

These equations are correctly known under two terms: the *discrete time Fourier series* or *discrete Fourier transform* (DFT).

Note that the sine and cosine frequencies in Equations 3.3 and 3.4 are related to m, the base frequency f_1 or the total time, or the sample interval and number of points, or the sample frequency and number of points. These four relationships are summarized mathematically as

$$f = mf_1 = \frac{m}{T_T} = \frac{m}{NT_s} = \frac{mf_s}{N} \tag{3.5}$$

For the continuous equations (Equations 3.1 and 3.2), m could go to infinity, but for the discrete equations (Equations 3.3 and 3.4), $m \leq N$. This is because when $m = N$, the period of sin $(2\pi Nf_1 nT_s)$ would be $(1/Nf1) = (1/f_s)$ that is one sample and a sine wave cannot be represented by less than one sample. Since the number of components cannot possibly exceed the number of points ($m \leq N$), the maximum theoretical frequency would be

$$f_{\text{theoretical max}} = mf_1 = Nf_1 = \frac{Nf_s}{N} = f_s \tag{3.6}$$

* These equations normalize by $2/T$, but other normalizations, such as $1/T$, are common. Unfortunately, there is no standard protocol for normalization. In the digital domain, normalization is usually not used as is the case here in Equations 3.3 and 3.4.

when $m = N$, $f = f_s$. However, as described in Section 1.6.2, there must be at least two sample points within a period to define a sine wave. To have two samples within a period, the maximum period with respect to sample interval is

$$2T_s = \frac{2}{f_s} = \frac{2}{Nf_1}$$

which corresponds to a maximum frequency of $(Nf_1/2)$. So, the maximum m must be $\leq N/2$. Hence, from Equation 3.5:

$$f_{\max} < \frac{Nf_1}{2} < \frac{Nf_s}{2N} < \frac{f_s}{2} \tag{3.7}$$

which is Shannon's sampling theorem. So, the maximum frequency component that can be obtained from a digitized signal is less than $f_s/2$ and occurs for all $m < N/2$. Recall that $f_s/2$ is termed the Nyquist frequency.

These equations are the most straightforward examples of the Fourier series analysis equations, but they are not the most commonly used. (The continuous time-domain equations, Equations 3.1 and 3.2, are used in calculations done manually, but such problems are only found in textbooks.) A more succinct equation that uses complex numbers is described later in this chapter. The basis used in Fourier series analysis consists of sines and cosines having frequencies only at discrete values of mf_1, which is either the same frequency as the waveform (when $m = 1$) or higher multiples (when $m > 1$) termed *harmonics*. Thus, the Fourier series analysis decomposes a waveform by projection on the basis of harmonically related sinusoids (or sines and cosines); so, the approach is sometimes referred to as *harmonic decomposition*.

Rather than plotting sine and cosine amplitudes obtained from these equations, it is more intuitive to plot the amplitude and phase angle of a single sinusoidal waveform such as $\cos(2\pi f + \theta)$. This is known as the polar rather than the rectangular representation of the harmonic components. The sine and cosine amplitudes can be converted into a single sinusoid using the well-known rectangular-to-polar equation:

$$a\cos(x) + b\sin(x) = C\cos(x - \theta) \tag{3.8}^*$$

The basic trigonometric identity for the difference of two arguments of a cosine function gives the conversion between the polar representation as a single cosine of magnitude C and angle $-\theta$, into a sine and cosine representation:

$$a = C\cos(\theta) \tag{3.9}$$

$$b = C\sin(\theta) \tag{3.10}$$

The reverse transformation, from sines and cosines to a signal sinusoid, is found algebraically from Equations 3.9 and 3.10 as

$$C = (a^2 + b^2)^{1/2} \tag{3.11}$$

$$\theta = \tan^{-1}\left(b/a\right) \tag{3.12}$$

Note that the θ has a negative sign in Equation 3.8; so, Equation 3.12 actually finds $-\theta$.

* Note that in engineering, it is common to specify the angle, θ, in deg, and the time component, x, in radians. Of course, MATLAB uses radians in all trigonometric functions, but conversion into deg is easy and is commonly done for phase curves.

EXAMPLE 3.1

Use Fourier series analysis to generate the magnitude and phase plot of the ECG signal originally shown in Figure 2.16 and repeated below. For these data, $f_s = 50$ Hz.

Solution

Fourier series analysis requires that the waveform must be periodic; so, we have to assume that the EEG signal is periodic. In other words, the EEG segment is assumed to be one cycle of a periodic signal as illustrated in Figure 3.5. Of course, this is beyond improbable, but it really does not matter since we know nothing about the signal before or after the segment on the computer; so, any assumption about that unknown signal is fine. Once this assumption is made, implementing Equations 3.3 and 3.4 in MATLAB is easy. Then we use Equations 3.11 and 3.12 to convert the sine and cosine components into magnitude and phase plots. We make $M = N/2$ since that corresponds to the maximum valid frequency to be found in the digitized signal (Equation 3.7). (Since this frequency is $f_s/2$ and $f_s = 50$ Hz, we anticipate a maximum frequency of 25 Hz.)

```
% Example 3.1 Use Fourier series analysis to generate the magnitude and
% phase plots of the ECG signal.
%
fs = 50;                    % Sample frequency
load eeg_data;              % Get data (vector eeg)
N = length(eeg);;           % Get N
Tt = N/fs;                  % Calculate total time
f1 = 1/Tt;                  % Calculate fundamental frequency
t = (1:N)/fs;               % Time vector for plotting
for m = 1:round(N/2)
  f(m) = m*f1;              % Sinusoidal frequencies
  a = sum(eeg.*cos(2*pi*f(m)*t));      % Cosine coeff., Eq. 3.3
  b = sum(eeg.*sin(2*pi*f(m)*t));      % Sine coeff., Eq. 3.4
```

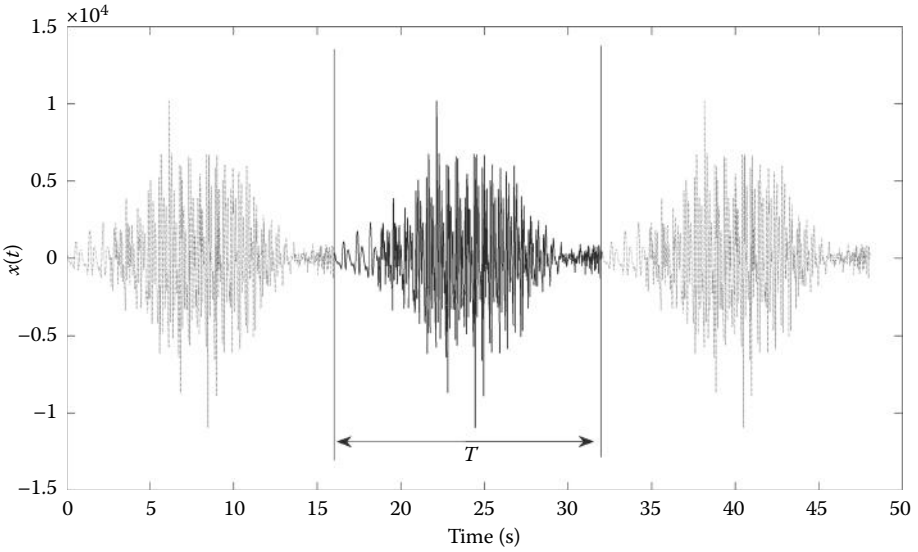

Figure 3.5 The EEG signal, originally shown in Figure 2.16, illustrated here as one period of a fictional periodic signal having period *T*.

```
  X_mag(m) = sqrt(a^2 + b^2);          % Magnitude spectrum Eq. 3.6
  X_phase(m) = -atan2(b,a);            % Phase spectrum, Eq. 3.7
end
X_phase = unwrap(X_phase);             % Compensates for shifts > pi
X_phase = X_phase*360/(2*pi);          % Convert phase to deg.
subplot(2,1,1);
plot(f,X_mag,"k");                     % Plot magnitude spectrum
     ..........Labels ........
subplot(2,1,2);
plot(f,X_phase,"k");                   % Plot phase spectrum
     ..........Labels ........
```

Analysis

The total time, T_T, of the signal was determined by dividing the number of points, N, by f_s (equivalent to multiplying N by T_s). The fundamental frequency, f_1, is $1/T_T$. A loop was used to correlate the sine and cosine waves with the EEG signal and their respective components were found by direct application of Equations 3.3 and 3.4. These components were converted into the more useful single sinusoidal representation of magnitude and phase using the transformation equations, Equations 3.6 and 3.7. Note that the component frequencies, mf_1, are saved in a vector for use in plotting.

While the magnitude spectrum is straightforward, determining the phase curve has some potential pitfalls. First, note that the negative arctangent is taken as θ as given in Equation 3.12. The next problem is in the implementation of the arctangent function. Many computer and calculator algorithms, including MATLAB's `atan` function, do not take into account the angle quadrant; so, for example, $\tan^{-1}(-b/a)$ gives the same result as $\tan^{-1}(b/-a)$, but the former has an angle between 0 and $-\pi/2$ whereas the latter has an angle between $\pi/2$ and π. MATLAB's `atan2` function checks the signs of both the numerator and denominator and adjusts the resultant angle accordingly.

```
angle = atan2(b,a); % Find the arctangent taking the quadrant in account
```

Another potential problem is that by definition, the phase angle is limited to $\pm\pi$ (or $0–2\pi$ but `atan2` returns $\pm\pi$). Yet, at some frequency, it is possible, in fact likely, that the phase component will exceed those limits. When this occurs, the phase curve will "wrap around" so that $\pi + \theta$ will have the value of $-\pi + \theta$. If we were dealing with single values, there would be little we can do, but since we are dealing with a curve, we might assume that large point-to-point transitions indicate that wrap around has occurred. One strategy would be to search for transitions greater than some threshold and, if found, reduce that transition by adding $+\pi$ or $-\pi$, whichever leads to the smaller transition. While it would be easy to write such a routine, it is unnecessary as MATLAB has already done it (yet again). The routine `unwrap` unwraps phase data in radians by changing absolute jumps $\geq\pi$ to their 2π complement.

```
phase_unwrapped = unwrap(phase);     % Unwrap phase data
```

After this routine is applied to the phase data, the phase is converted into deg that are more conventionally used in engineering. The magnitude and phase curves are then plotted as functions of frequency.

Results

The plots generated in this example are shown in Figure 3.6. The magnitude plot is roughly similar to the plot in Figure 2.16 produced in Example 2.10 by taking the maximum cross-correlation between sines and the signal. However, in that example, the sinusoidal frequencies

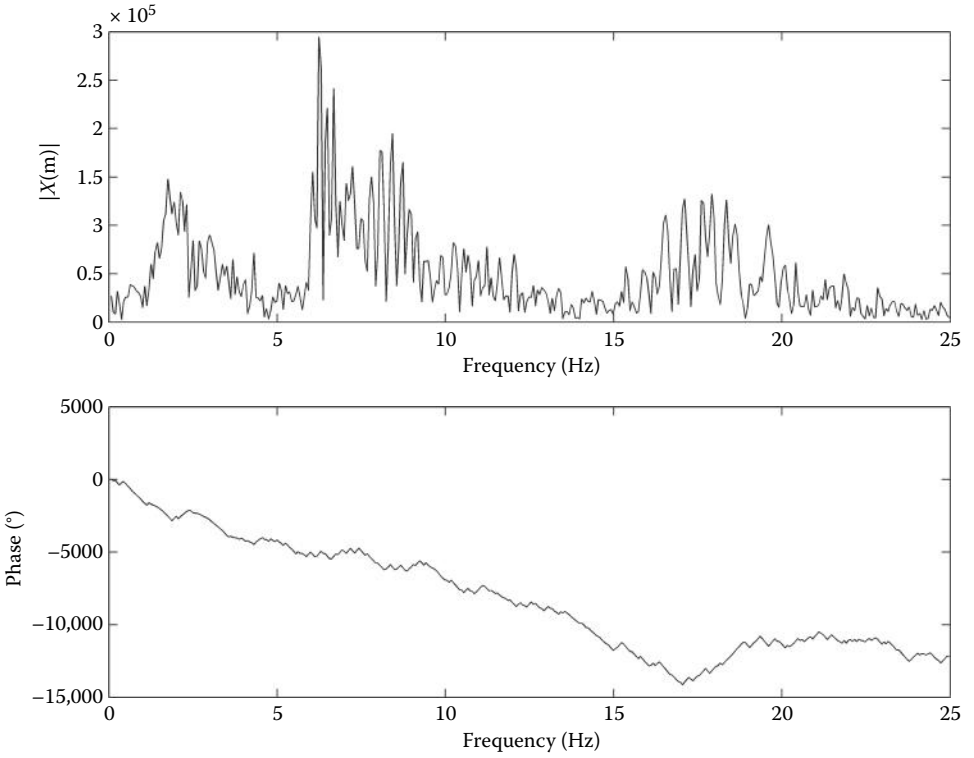

Figure 3.6 The magnitude and phase spectrum of the EEG signal shown in Figure 3.5 as determined by projecting the EEG signal on a basis of harmonically related sines and cosines ($f = mf_1$ where $m = 1,2,3,\ldots$). Special attention is paid to the construction of the phase plot. The magnitude spectrum is also shown in Figure 3.1.

were selected arbitrarily (0.25–25 Hz in increments of 0.25 Hz) and were not related to the fundamental frequency of the EEG signal. If we check the values of `Tt` and `f1` in this example, we find the total time of EEG is 16.02 s and the related fundamental frequency is 0.0624 Hz. So, a major advantage in using Fourier series decomposition is that it tells you exactly what sinusoidal frequencies might exist in the waveform: specifically, the fundamental frequency, f_1, and higher harmonics, mf_1. This assumes that the signal is periodic but, as stated above, that assumption must be made to apply Fourier analysis to a digital signal.

There is another advantage in using Fourier series analysis as opposed to probing with sinusoids at arbitrary frequencies. The code in Example 3.1 is much faster than the code in Example 2.10. In fact, it could be made even faster by using a special algorithm that employs complex numbers and breaks the signal into smaller time segments. This fast algorithm is known, predictably, as the *fast Fourier transform* or commonly as the FFT.

The Fourier series analysis is part of an invertable transform: Fourier analysis converts a time function into the frequency domain as equivalent sinusoidal components and the frequency-domain Fourier sinusoidal components can be summed to reconstruct the time-domain waveform. In the continuous domain, the equations become:

$$x(t) = \frac{a(0)}{2} + \sum_{m=1}^{\infty} a[m]\cos(2\pi m f_1 t) + \sum_{m=1}^{\infty} b[m]\sin(2\pi m f_1 t) \tag{3.13}$$

where the $a[m]$ and $b[m]$ are defined in Equations 3.1 and 3.2 and the $a(0)$ component is used to add in any nonzero DC component. The DC component is evaluated as twice the mean of the $x(t)$:

$$a(0) = \frac{2}{T}\int_0^T x(t)dt \tag{3.14}$$

The normalization is by $2/T$ to correspond to that used in Equations 3.1 and 3.2. Note that the "2" is divided back in Equations 3.13 and 3.15. Using the transformation equations given in Equations 3.11 and 3.12, the harmonic series in Equation 3.13 can also be expressed in terms of a single sinusoid family with members having different amplitudes and phases:

$$x(t) = \frac{a(0)}{2} + \sum_{m=1}^{\infty} C(m)\cos(2\pi m f_1 t + \theta(m)) \tag{3.15}$$

The operation represented by Equation 3.13 or Equation 3.15 is often termed the *inverse Fourier transform*, particularly when referring to the discrete operations. A discrete version of these equations is presented in Section 3.2.2.

Most simple waveforms can be well approximated by only a few family members. This is illustrated in the next example that decomposes a triangular waveform and then reconstructs it with limited components.

EXAMPLE 3.2

Perform a discrete Fourier series analysis on the triangular waveform defined by the equation:

$$x(t) = \begin{cases} t & 0 \le t < 0.5 \text{ s} \\ 0 & 0.5 \le t < 1.0 \text{ s} \end{cases}$$

Reconstruct the waveform using the first five and then the first 10 components. Plot the reconstructed waveforms superimposed on $x(t)$. Make $f_s = 500$ Hz.

Solution

After constructing the waveform in MATLAB, decompose the waveform into 10 components using the approach featured in Example 3.1. Note that the total time of the waveform is 1.0 s. Then reconstruct the two waveforms using the digital version of Equation 3.15. However, if we use Equation 3.15 for reconstruction, then we need to normalize in a manner equivalent to that used in Equations 3.1 and 3.2, that is, by $2/N$. Note that the waveform does have a DC term that should be added to the reconstructions.

```
% Example 3.2 Fourier series decomposition and reconstruction.
%
fs = 500;                  % Sampling frequency
Tt = 1;                    % Total time
N = Tt*fs;                 % Determine N
f1 = 1/Tt;                 % Fundamental frequency
t = (1:N)/fs;              % Time vector
x = zeros(1,N);            % Construct waveform
x(1:N/2) = t(1:N/2);
% Fourier decomposition
a0 = 2*mean(x);            % Calculate a(0)
```

```
for m=1:10
   f(m)=m*f1;           % Sinusoidal frequencies
   a= (2/N)*sum(x.*cos(2*pi*f(m)*t));        % Cosine coeff., Eq. 3.3
   b=(2/N)*sum(x.*sin(2*pi*f(m)*t));         % Sine coeff., Eq. 3.4
   X_mag(m)=sqrt(a^2+b^2);                   % Magnitude spectrum Eq. 3.6
   X_phase(m)=-atan2(b,a);                   % Phase spectrum, Eq. 3.7
end
x1=zeros(1,N);              % Reconstruct waveform
for m=1:5
   f(m)=m*f1;               % Sinusoidal frequencies
   x1=x1+X_mag(m)*cos(2*pi*f(m)*t+X_phase(m)); % Eq. 3.15
end
x1=x1+a0/2;                 % Add in DC term
subplot(2,1,1);
plot(t,x1,'k'); hold on;
plot(t,x,'-k');             % Plot reconstructed and original waveform
     .........labels and title.........
    .........Repeat for 10 components.........
```

Results

Figure 3.7 shows the original triangular waveform and the reconstructed waveforms based on only five or 10 components. The reconstruction using 10 components is reasonably close, but shows oscillations. Such oscillations will occur whenever there is a discontinuity in the waveform and a limited number of components is used to reconstruct the waveform.

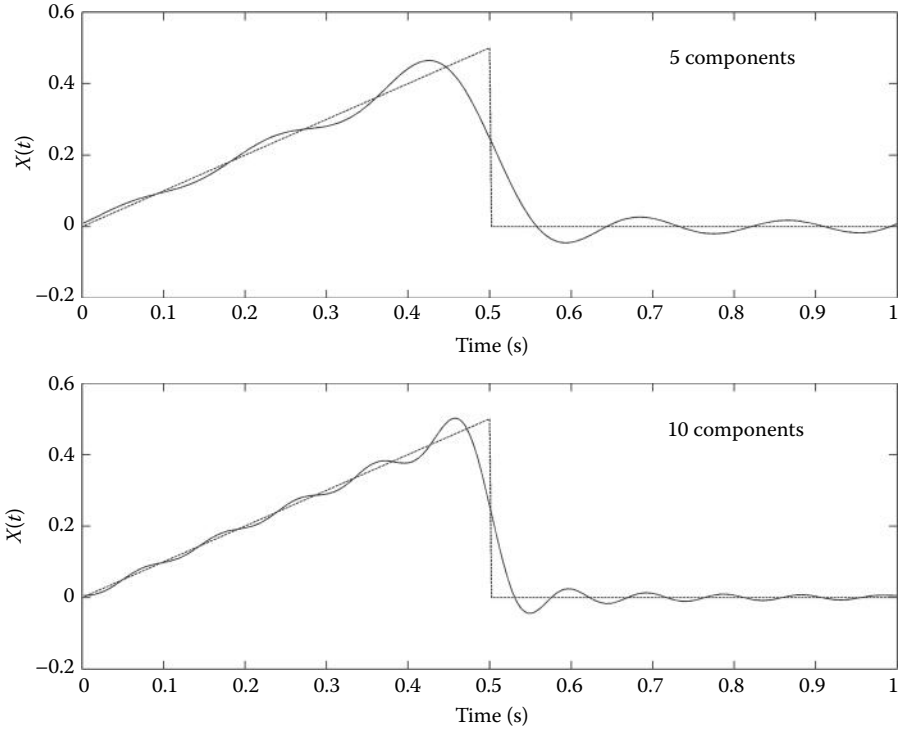

Figure 3.7 A triangular waveform (dashed lines) decomposed using the Fourier series analysis and reconstructed using only 5 or 10 components plus the DC term. The oscillations, known as the Gibbs oscillations, occur when limited components are used to reconstruct a waveform.

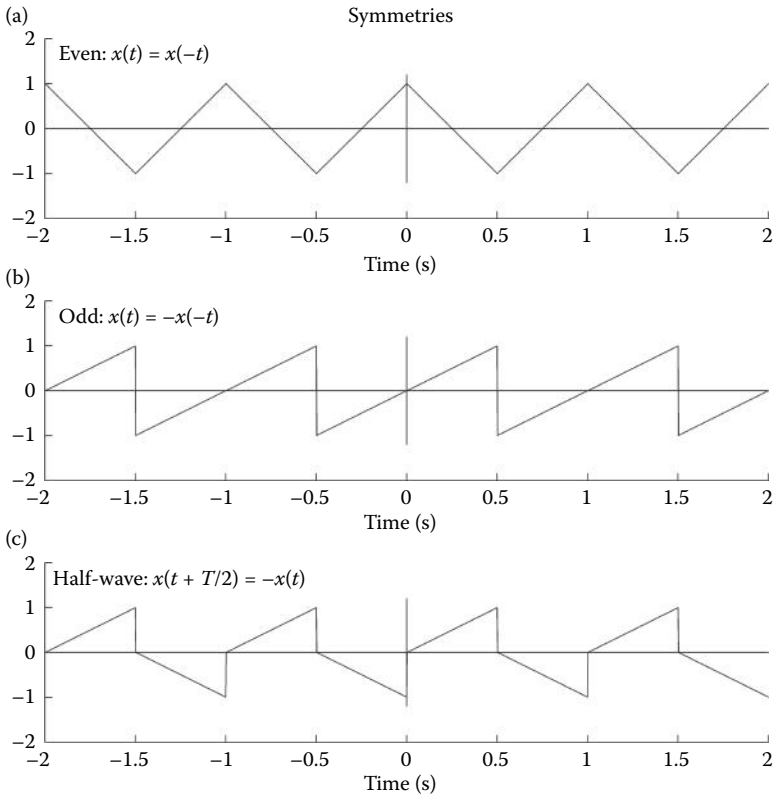

Figure 3.8 Three waveforms illustrating symmetries useful in the Fourier series analysis. Each waveform has a period of 1.0 s. (a) Even symmetry, (b) odd symmetry, and (c) half-wave symmetry.

3.2.1.1 Symmetry

Some waveforms are symmetrical or antisymmetrical about $t = 0$, so that one or the other of the components, a_m or b_m in Equations 3.1 and 3.2, will be zero. Specifically, if the waveform has mirror symmetry about $t = 0$, that is, $x(t) = x(-t)$ (Figure 3.8a), then multiplications with all sine functions will integrate or sum to zero so that the b_m terms will be zero. Such mirror symmetry functions are termed *even* functions. If the function has antisymmetry, $x(t) = -x(t)$, (Figure 3.8b), it is termed an *odd* function and all multiplications with cosines will integrate or sum to zero so that the a_m coefficients will be zero. Finally, functions that have *half-wave* symmetry will have no even coefficients; so, both a_m and b_m will be zero for even m. These are functions where the second half of the period looks like the first half, but is inverted, that is, $x(t + T/2) = -x(t)$ (Figure 3.8c). These functions can be thought of as having the first half shifted to become the second half and then inverted. Functions having half-wave symmetry are also even functions. These symmetries are useful for simplifying the task of solving for the coefficients manually, but are also useful for checking solutions done on a computer. Table 3.1 and Figure 3.8 summarize these properties.

3.2.2 Complex Representation

The equations for computing the Fourier series analysis of digitized data are usually presented using complex variables, as this leads to more succinct equations. In addition, the FFT is coded using this representation. By using complex variables notation, the sine and cosine terms of Equations 3.3 and 3.4 can be represented by a single exponential term using Euler's identity:

$$e^{jx} = \cos x + j \sin x \tag{3.16}$$

Table 3.1 Function Symmetries		
Function Name	**Symmetry**	**Coefficient Values**
Even	$x(t) = x(-t)$	$b[m] = 0$
Odd	$x(t) = -x(-t)$	$a[m] = 0$
Half-wave	$x(t) = -x(t+T/2)$	$a[m] = b[m] = 0$; for m even

(Note: Mathematicians use "i" to represent $\sqrt{-1}$, whereas engineers use "j" since "i" is reserved for current.) Using complex notation, the equation for the continuous FT can be derived from Equations 3.1, 3.2, and 3.16 using only algebra:

$$X(f) = \int_0^T x(t)e^{-j2\pi ft}dt \quad f = -\infty,\ldots -2f_1, -f_1, 0, f_1, 2f_1,\ldots,\infty \tag{3.17}$$

where the variables are defined as in Equations 3.1 and 3.2. Again, the frequency is obtained from the value of m, that is, $f = mf_1$ where $f_1 = 1/T$. In discrete form, the integration becomes summation and $t \rightarrow nT_s$

$$X[m] = \sum_{n=1}^{N} x[n]e^{(-j2\pi mf_1 nT_s)} \quad m = -N/2,\ldots, -1, 0, 1, \ldots N/2 \tag{3.18}$$

This is the complex form of the DFT or discrete time Fourier series. An alternative equation can be obtained by substituting $f_1 = 1/T = 1/NT_s$ for f_1 in Equation 3.18:

$$X[m] = \sum_{n=1}^{N} x[n]e^{(-j2\pi mnT_s/NT_s)} = \sum_{n=1}^{N} x[n]e^{(-j2\pi mn/N)} \tag{3.19}$$

This is the common form of the discrete Fourier series analysis equations; it is the format found in most publications that use this analysis. The family number, m, must now be allowed to be both positive and negative when used in complex notation: $m = -N/2, \ldots, N/2-1$. Although the equations look different, the game is the same: analysis by projecting the waveform ($x[n]$) on a harmonic sinusoidal basis, only the basis is now represented using complex variables. It is common to use capital letters for frequency-domain variables as in Equations 3.18 and 3.19.

The inverse FT can be calculated as

$$x[n] = \frac{1}{N}\sum_{n=1}^{N} X[n]e^{(-j2\pi mn/N)} \tag{3.20}$$

Applying Euler's identity (Equation 3.16) to the expression for $X[m]$ in Equation 3.20 gives

$$X[m] = \sum_{n=1}^{N} x[n]\cos[2\pi mn/N] + j\sum_{n=1}^{N} x[n]\sin[2\pi mn/N] \tag{3.21}$$

Transforming this equation into polar form using the rectangular-to-polar transformation described in Equations 3.11 and 3.12, it can be shown that $|X[m]|$ equals $\sqrt{a[m]^2 + b[m]^2}$ (the magnitude for the sinusoidal representation of the Fourier series), whereas the angle of $X[m]$ that is $\tan^{-1}(jb[m]/a[m])$ (the phase angle for this representation of the sinusoid).

As mentioned above, $X[m]$ must be allowed to have both positive and negative values for m for computational reasons. Negative values of m imply negative frequencies, but these are only

a computational necessity and have no physical meaning. In some versions of the Fourier series equations shown above, Equation 3.19 is multiplied by T_s (the sampling time) whereas Equation 3.20 is divided by T_s so that the sampling interval is incorporated explicitly into the Fourier series coefficients. Other methods of scaling these equations can be found in the literature.

Originally, the FT (or Fourier series analysis) was implemented by direct application of the above equations, usually using the complex formulation. These days, the FT is implemented by a more computationally efficient algorithm, the FFT, which cuts the number of computations from N^2 to 2 log N, where N is the length of the digital data. For large N, this is a substantial improvement in processor time. The implementation of the basic FT in MATLAB is

```
X = fft(x,N)
```

where `x` is the input waveform and `X` is a complex vector providing the complex sinusoidal coefficients. The argument `n` is optional, but is often quite useful. It is used to modify the length of the signal to be analyzed: if `N < length(x),` then the signal that is analyzed is truncated to the first `N` points; if `N > length(x), x` is padded with trailing zeros to equal `n`. The length of the complex output vector, `X`, is the same length as the adjusted (padded or truncated) length of `x`; however, only the first half of the points are valid. If the signal is real, the second half of the magnitude spectrum of `X` is just the mirror image of the first half, and the second half of the phase spectrum is the mirror image inverted. So, the second half of both the magnitude and phase spectrum contains redundant information; however, note that the number of nonredundant points in the spectrum is the same as in the original signal, since essentially, two vectors are generated, magnitude and phase, each with half the valid number of points as the original signal. This "conservation of samples" is essential, since the FT is *invertable* and we must be able to reconstruct the original waveform from its spectrum.

The `fft` routine implements Equation 3.19 above using a special high-speed algorithm. Calculation time is highly dependent on data length and is fastest if the data length is a power of two, or if the length has many prime factors. For example, on one machine, a 4096-point FFT takes 2.1 s, but requires 7 s if the sequence is 4095-points long and 58 s if the sequence is 4097 points. If at all possible, it is best to stick with data lengths that are powers of two.

The output of the `fft` routine is a complex vector in which the real part corresponds to $a[m]$ in Equation 3.3 and the imaginary part corresponds to $b[m]$ in Equation 3.4. The first entry $X[1]$ is always real and is the DC component; thus, the first frequency component is the complex number in $X[2]$. It is easy to convert this complex number, or the entire complex vector, into polar form (i.e., magnitude and phase). To get the magnitude, take the absolute value function, `abs`, of the complex output `X`:

```
Magnitude = abs(X)
```

This MATLAB function simply takes the square root of the sum of the real part of `X` squared and the imaginary part of `X` squared. As with many MATLAB routines, `X` can be either a scalar or a vector and either a scalar or a vector is produced as the output. The phase angle can be obtained by using the MATLAB `angle` function:

```
Phase = angle(X)
```

The `angle` function takes the arctangent of the imaginary part divided by the real part of `X`. The output is in radians and should be converted into deg if desired.

The inverse FT (Equation 3.20) is implemented in MATLAB using the routine `ifft`. The calling structure is

```
x = ifft(X,N);          % Take the inverse Fourier transform
```

An example applying the MATLAB's `fft` to an array containing sinusoids and white noise is given next. This example uses a special routine, `sig _ noise`, found in the accompanying material. The routine is used throughout this book and generates data consisting of sinusoids and noise useful in evaluating spectral analysis algorithms. The calling structure for `sig_ noise` used in this example is

```
[x,t] = sig_noise([f],[SNR],N);
```

where `f` specifies the frequency of the sinusoid(s) in Hz, `N` is the number of points, and `SNR` specifies the desired noise in dB associated with the sinusoid(s). The routine assumes a sample frequency of 1 kHz. If `f` is a vector, multiple sinusoids are generated. If `SNR` is a vector, it specifies the SNR for each frequency in f; if it is a scalar, then all sinusoids generated have that SNR. The output waveform is in vector `x`; `t` is a time vector useful in plotting.

EXAMPLE 3.3

Plot the magnitude and phase spectrum of a waveform consisting of a single sine wave and white noise with an SNR of –7 dB. Make the waveform 1024-points long. Use MATLAB's `fft` routine and plot only the valid points.

Solution

Specify N (1024) and f_s (1000 Hz) and generate the noisy waveform using `sig _ noise`. Generate a frequency vector for use in plotting. This vector should be the same length as the signal and ranges to f_s (recall that when $m = N, f = f_s$, from Equations 3.5 and 3.6). Take the FT using `fft` and calculate the magnitude and phase from the complex output. Unwrap the phase and convert it into deg. Plot only the valid points from 1 to $N/2$. Since the first element produced by the `fft` routine is the DC component, the frequency vector should range between 0 and N–1 if the DC term is included in the spectral plot.

```
% Example 3.3 Determine spectrum of a noisy waveform
%N = 1024;                      % Data length
N2 = 511;                       % Valid spectral points
fs = 1000;                      % Sample frequency (assumed by sig_noise)
[x,t] = sig_noise(250,-7,N);    % Generate signal (250 Hz sine plus white noise)
X = fft(x);                     % Calculate FFT
X_mag = abs(X);                 % Compute magnitude spectrum
Phase = unwrap(angle(X));       % Phase spectrum unwraped
Phase = Phase*360/(2*pi);       % Convert phase to deg
f = (0:N-1)*fs/N;               % Frequency vector
subplot(2,1,1);
plot(f(1:N2),X_mag(1:N2),'k');      % Plot magnitude spectrum
  ......Labels and title.......
subplot(2,1,2);
plot(f(1:N2),Phase(1:N2),'k'        % Plot phase spectrumy
label('Phase (deg)','FontSize',14);
  ......Labels and title.......
```

Results

The resulting plots are shown in Figure 3.9. The presence of a 250-Hz sine wave is easy to identify even though it is "buried" in considerable noise.

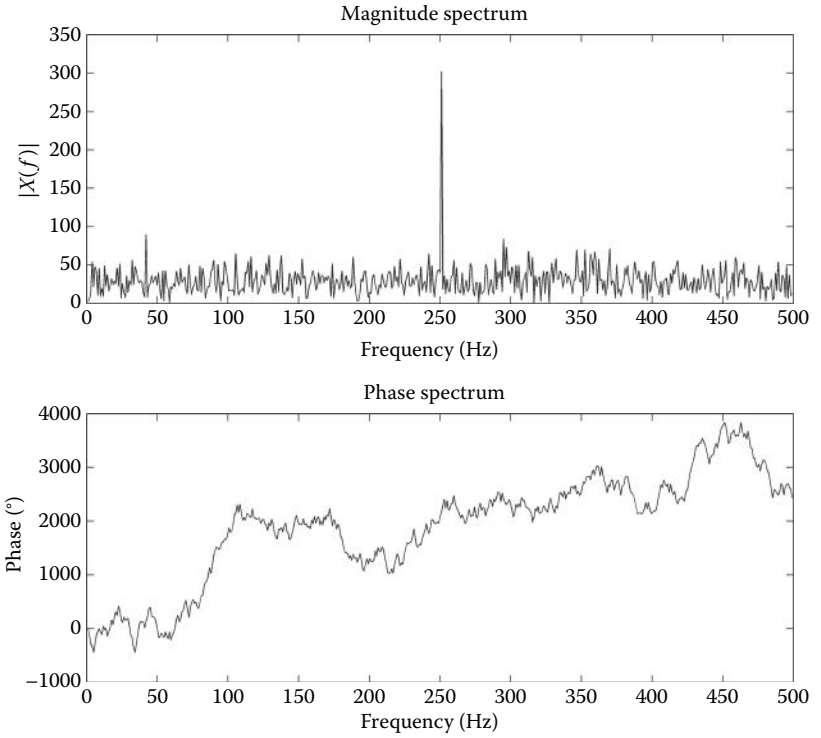

Figure 3.9 Magnitude and phase spectral plots produced by the program in Example 3.3. The peak at 250 Hz is apparent. The sampling frequency of these data is 1 kHz; hence, the spectrum is valid up to the Nyquist frequency, $f_s/2$ (500 Hz). Accordingly, only the first half of the vector produced by `fft` is plotted. (SNR = –7 dB, $N = 1024$.)

3.2.3 Data Length and Spectral Resolution

For the DFT, the length of the data, N, and the sample time, T_s (or f_s), completely determine the frequency range of the spectrum since $f_{\min} = f_1 = 1/T = 1/(NT_s)$ and $f_{\max} < f_s/2 = 1/2T_s$. So, in terms of T_s:

$$\frac{1}{NT_s} \leq f < \frac{1}{2T_s} \tag{3.22}$$

Or, using f_s:

$$\frac{f_s}{N_s} \leq f < \frac{f_s}{2} \tag{3.23}$$

These two variables, N and T_s, also determine the frequency resolution of the spectrum. Since the spectral frequencies are at $f = mf_1$ (Equation 3.5), the spectral frequencies are separated by f_1 that is equal to

$$f_{\text{Resolution}} = f_1 = \frac{1}{T_T} = \frac{1}{NT_s} = \frac{f_s}{N} \tag{3.24}$$

For a given sampling frequency, the larger the number of samples in the signal, N, the smaller the frequency increment between successive DFT data points: the more points sampled, the higher the spectral resolution.

Once the data have been acquired, it would seem that the number of points representing the data, *N*, is fixed, but there is a trick that can be used to increase the data length post hoc. We can increase *N* simply by tacking on constant values, usually zeros, that is, zero padding. This may sound like cheating, but it is justified by the underlying assumption that the signal is actually unknown outside the data segment on the computer. Zero padding gives the appearance of a spectrum with higher resolution since it decreases the distance between frequency points and the more closely spaced frequency points show more of the spectrum's details. In fact, extending the period with zeros does *not* increase the information in the signal and the resolution of the signal is really not any better. What it does is to provide an interpolation between the points in the unpadded signal: it fills in the gaps of the original spectrum using an estimation process. Overstating the value of zero padding is a common mistake of practicing engineers. Zero padding does not increase the resolution of the spectrum, only the apparent resolution. However, the interpolated spectrum will certainly look better when plotted.

EXAMPLE 3.4

Generate a 0.5-s symmetrical triangle wave assuming a sample frequency of 100 Hz so that $N = 50$ points. Calculate and plot the magnitude spectrum. Zero pad the signal so that the period is extended to 2 and 8 s and calculate and plot the new magnitude spectra. Limit the spectral plot to a range of 0–10 Hz.

Solution

First, generate the symmetrical triangle waveform. Since $f_s = 100$ Hz, the signal should be padded by 150 and 750 additional samples to extend the 0.5-s signal to 2.0 and 8.0 s. Calculate the appropriate time and frequency vectors based on the padded length. Calculate the spectrum using `fft` and take the absolute value of the complex output to get the magnitude spectra. Use a loop to calculate and plot the spectra of the three signals. As in Example 3.3, the frequency vector should range from 0 to *N*–1 since the DC term is included in the spectral plots.

```
% Example 3.4 Calculate the magnitude spectrum of a simple waveform with
% and without zero padding.
%
fs = 100;                          % Sample frequencies
N1 = [0 150 750];                  % Padding added to signal
x =[(0:25) (24:-1:0)];             % Generate basic test signal
for k = 1:3
    x1 = [x zeros(1,N1(k))];       % Zero pad signal
    N = length(x1);                % Data length
    t = (1:N)/fs;                  % Time vector for plotting
    f = (0:N-1)*fs/N;              % Frequency vector for plotting
    subplot(3,2,k*2-1);
    plot(t,x1,"k");                % Plot test signal
       .......Label and title.......
    xlim([0 t(end)]);
    subplot(3,2,k*2);
    X1 = abs(fft(x1));             % Calculate the magnitude spectrum
    plot(f, X1,".k");              % Plot magnitude spectrum
    axis([0 10 0 max(X1)*1.2]);
       .......Label and title.......
end
```

Results

The time and magnitude spectrum plots are shown in Figure 3.10. All the spectral plots have the same shape, but the points are more closely spaced with the zero-padded data. Simply adding

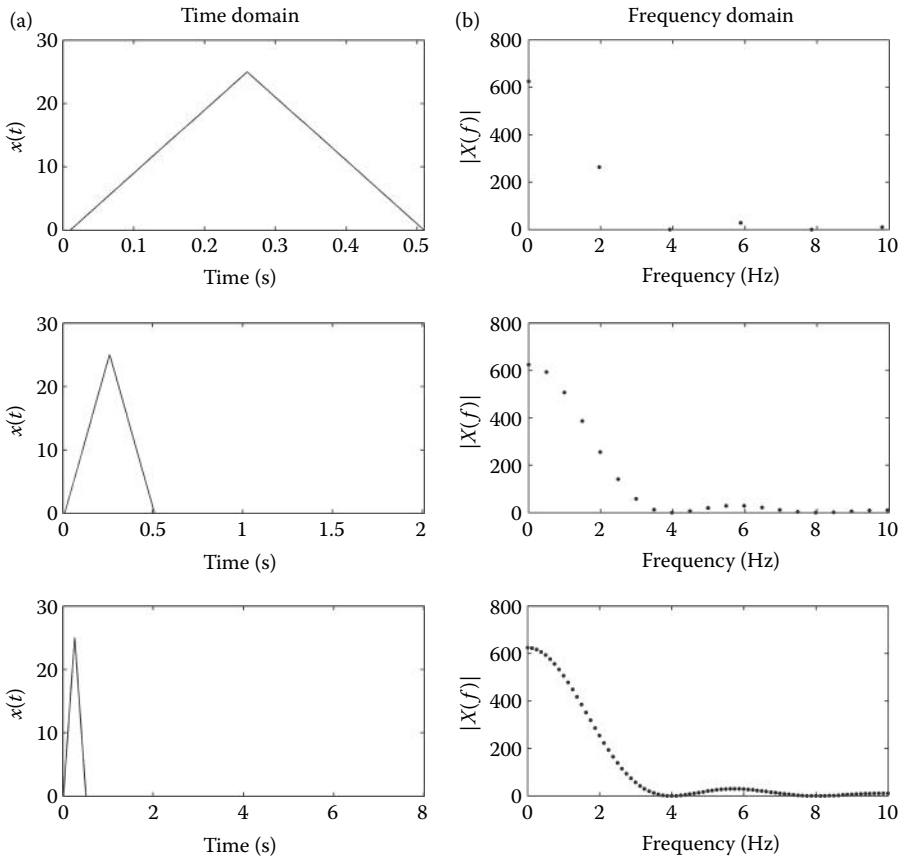

Figure 3.10 A triangular waveform padded with zeros to three different periods. Upper: $T_T = 0.5$ s (no padding). Middle: $T_T = 2.0$ s (padded with 150 zeros). Lower: $T_T = 8.0$ s (padded with 750 zeros).

zeros to the original signal produces a better-looking curve, which explains the popularity of zero padding even if no additional information is produced. In this example, the signal was padded before calling the `fft` routine since we wanted time plots of the padded signals. Otherwise, the second argument of the `fft` routine could have been used to pad the signal as described above. Also, the entire spectrum was plotted including the redundant points, but these points were eliminated by limiting the x-axis using `xlim`.

3.2.3.1 Aperiodic Functions

As mentioned above, the DFT makes the assumption that the signal is periodic with a period of $T_T = NT_s$, and the frequency spectrum produced is a set of individual numbers spaced f_s/N apart (Equation 3.24). To emphasize the individual nature of the spectral points, sometimes, the points are plotted as lines from the horizontal axis to the point value as shown in Figure 3.11a. This gives rise to the term *line spectra* for the spectra determined from periodic functions.

The frequency of the spectral point is $f = mf_1 = mf_s/N$ since $f_1 = (1/NT_s) = (f_s/N)$. As the period, NT_s, gets longer, N would increase and the distance between the spectral points, f_s/N, would get closer together. The longest period imaginable would be infinite, $T_T = NT_s \to \infty$, which would require an infinite number of points, $N \to \infty$. As the period becomes infinite, the spacing between points, $f_s/N \to 0$, and m/N goes to a continuous variable f. Equation 3.19 becomes, theoretically, a function of the continuous variable f:

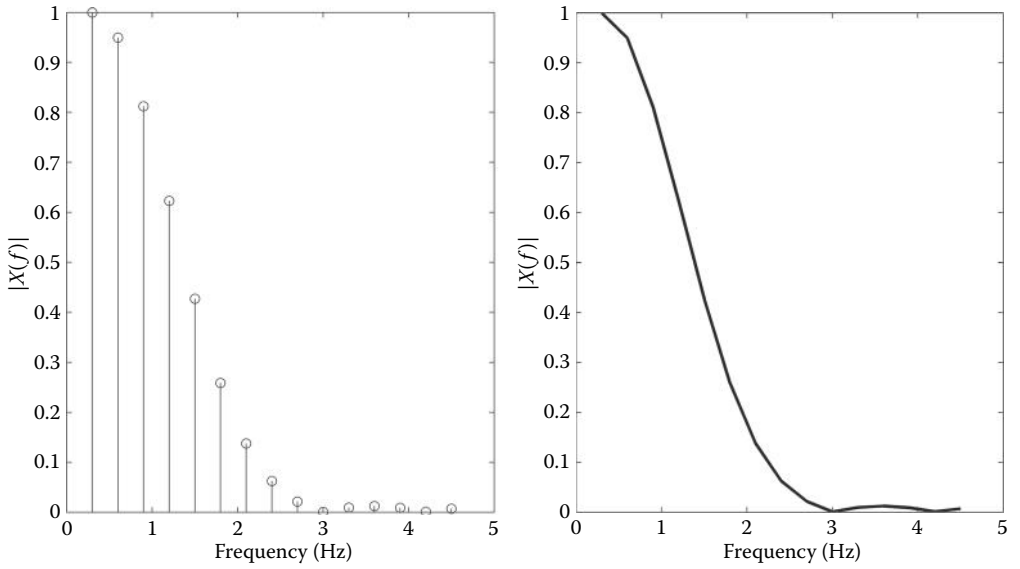

Figure 3.11 (a) The spectrum calculated by the discrete time Fourier series consists of individual points spaced f_s/N apart. (Recall that $f_1 = 1/(NT_s) = f_s/N$ and is equal to 0.3 Hz in this figure.) Sometimes, these individual points are plotted using vertical lines as shown, giving rise to the term *line spectra* for these plots. (b) If we assume that the original waveform is really aperiodic, then the distance between lines theoretically goes to 0 and the points can be connected.

$$X[f] = \sum_{n=1\infty}^{\infty} x[n]e^{-j2\pi nf} \tag{3.25}$$

Such signals are termed *aperiodic*: they are nonzero only for a well-defined period of time and are zero everywhere else. For analog signals, it is possible to calculate the FT of an aperiodic signal. This is termed the *continuous time FT* (as opposed to the Fourier series for periodic signals). The discrete version of the FT for aperiodic signals is termed the *discrete time Fourier transform* (DTFT). This operation has only theoretical significance, since a discrete aperiodic signal requires an infinite number of points and cannot exist in a real computer. However, if we pretend that the signal was actually aperiodic, we can plot the spectrum not as a series of points, but as a smooth curve (Figure 3.11b). Although this is commonly done when plotting the spectra, you should be aware of the underlying truth: an implicit assumption is being made that the signal is aperiodic; the calculated spectrum is really a series of discrete points that have been joined together because of this assumption. Again, true aperiodic signals are not possible in a real computer and Equation 3.25 is only of theoretical interest. It "does not compute" on a digital computer.

In summary, there are four versions of "Fourier transform": versions for periodic and aperiodic signals and for continuous and discrete signals. For each of these, there is a transform and an inverse transform. The equations for these four transforms are shown in Table 3.2 and the equations for the four inverse transforms are shown in Table 3.3. Note that in the rest of this book, only the DFT (also known as the discrete time Fourier series) and its inverse are used.

3.2.4 Window Functions: Data Truncation

The digitized versions of real biosignals are usually segments of a much longer, possibly infinite, time series. Examples are found in EEG and ECG analysis where the waveforms being analyzed continue over the lifetime of the subject. Obviously, only a portion of such waveforms can be

Table 3.2 Analysis Equations (Forward Transform)[a]

$X(t)$ periodic and continuous: Fourier series: $X[m] = \frac{1}{T}\int_0^T x(t)e^{-j2\pi m f_1 t}dt \quad m = 0, \pm 1, \pm 2, \pm 3, \ldots$ $f_1 = 1/T$	$x[n]$ periodic and discrete: tDFT or discrete time Fourier series[b,c]: $X[m] = \sum_{n=1}^{N} X[n]e^{-j2\pi mn/N}$ $m = 0, \pm 1, \pm 2, \pm \ldots \pm N/2$
$x(t)$ aperiodic and continuous: FT or continuous time FT: $X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}\,dt$	$x[n]$ aperiodic and discrete: DTFT[d]: $X(f) = \sum_{n=-\infty}^{\infty} x[n]e^{-j2\pi nf}$

[a] Often, ω is usually substituted for $2\pi f$ to make the equations shorter. In such cases, the Fourier series integration is carried out between 0 and 2π or from $-\pi$ to $+\pi$.
[b] Alternatively, $2\pi mnT/N$ where T is the sampling interval is sometimes used instead of $2\pi mn/N$.
[c] Sometimes, this equation is normalized by $1/N$ and then no normalization term is used in the inverse DFT.
[d] The DTFT cannot be calculated on a computer since the summations are infinite. It is used in theoretical problems as an alternative to the DFT.

represented in the finite memory of the computer, and some attention must be paid to how the waveform is truncated. Often, a segment is simply cut out from the overall waveform, that is, a portion of the waveform is truncated and stored, without modification, in the computer. This is equivalent to the application of a rectangular *window* to the overall waveform and the analysis is restricted to the *windowed* portion of the waveform. The window function, $w(t)$, for a rectangular window is simply 1.0 over the length of the window and 0.0 elsewhere (Figure 3.12, middle trace). Note that a rectangular window will usually produce abrupt changes or discontinuities at the two endpoints (Figure 3.12, lower trace). Window shapes other than rectangles are possible simply by multiplying the waveform by the desired shape (sometimes, these shapes are referred to as "tapering" functions). (See Section 2.3.1.3 and Figure 2.11.)

Table 3.3 Synthesis Equations (Reverse Transform)

Fourier Series: $x(t) = \sum_{n=-\infty}^{\infty} X[m]e^{j2\pi mf_1 t} \quad m = 0, \pm 1, \pm 2, \pm 3, \ldots$ $m = 0, \pm 1, \pm 2, \pm \ldots \pm N/2$	DFT or discrete time Fourier series[a,b]: $x[n] = \frac{1}{N}\sum_{n=1}^{N} X[m]e^{j2\pi mn/N}$ $m = 0, \pm 1, \pm 2, \pm \ldots \pm N/2$
FT or continuous time FT: $x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft}\,df$	DTFT[c,d]: $x[n] = \frac{1}{T}\int_0^T X(f)e^{j2\pi nft}\,df$

[a] Alternatively, $2\pi mnT/N$ where T is the sampling interval is sometimes used instead of $2\pi mn/N$.
[b] Sometimes, this equation is normalized by $1/N$ and then no normalization term is used in the inverse DFT.
[c] The DTFT cannot be calculated on a computer since the summations are infinite. It is used in theoretical problems as an alternative to the DFT.
[d] Requires an integral because the summations that produce $X(f)$ are infinite in number.

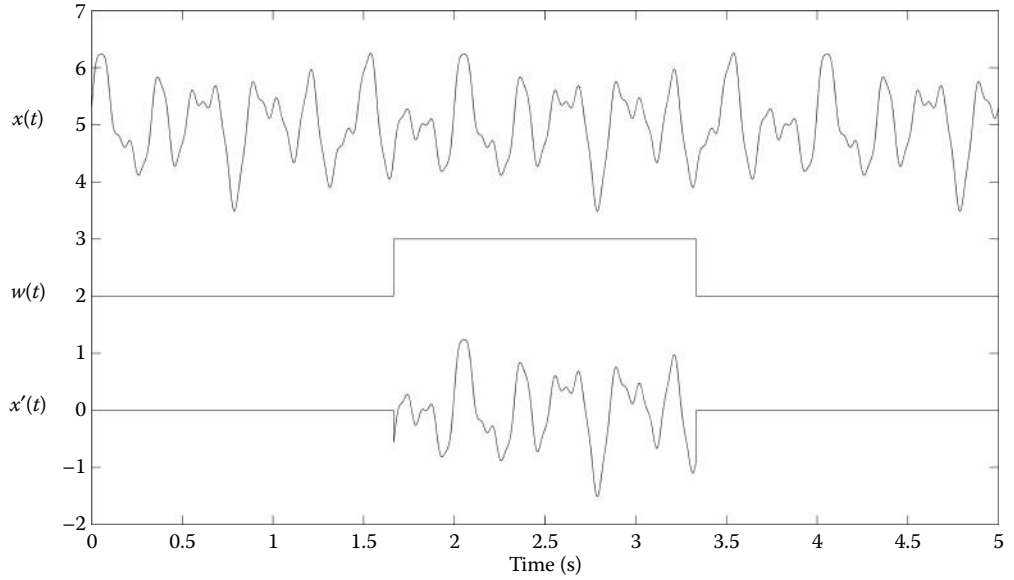

Figure 3.12 If a waveform is simply cut at the beginning and end of the portion stored in the computer, it is mathematically equivalent to multiplying the waveform, $x(t)$ by a rectangular window function, $w(t)$. This usually results in a discontinuity at the endpoints.

The act of windowing has an effect on a signal's spectrum, even if a rectangular window (i.e., simple truncation) is used. Mathematically, windowing is the multiplication of the signal with a window function. Multiplication in the time domain is equivalent to convolution in the frequency domain (and vice versa); so, the original spectrum is convolved with the spectrum of the window. Ideally, we would like this convolution to leave the signal spectrum unaltered. This will occur only if the spectrum of the window function is a unit impulse, that is, $W[f] = 1.0$ for $f = 0$ and 0.0 for all other frequencies. So, an idea of the artifact produced by a given window can be obtained from the FT of the window function itself. The deviations of the window's spectrum from a true impulse function show how it will modify the signal's spectrum.

The spectra of three popular windows are shown in Figure 3.13. The rectangular window spectrum (Figure 3.13a) is not exactly an impulse, although it does have a narrow peak at $f = 0$ Hz about 15 dB above the background peaks at $x(t)$. This peak around 0.0 Hz is called the *mainlobe*. The finite width of the mainlobe means that when it is convolved with the spectrum of the original signal, frequencies in the windowed spectrum will include an average of nearby frequencies in the original spectrum. Thus, peaks in a windowed spectrum will not be as precisely defined as when no window is used (i.e., the signal was of infinite length). Of more concern are the fairly large multiple peaks beginning at –15 dB and dropping to around –35 dB (Figure 3.13a). These are termed *sidelobes*, and they indicate the amount of distant frequencies that are merged into the windowed spectrum. They are of particular concern when a large spectral peak is near to a much smaller peak. Because of the merging produced by the side lobes, the smaller peak may be masked by the larger peak.

Other window shapes can be imposed on the data by multiplying the truncated waveform by the desired shape. There are many different window shapes and all have the effect of tapering the two ends of the data set toward zero. An example of a tapering window is the *Hamming window*, shown in Figure 3.14. This window is often used by MATLAB as a default window in routines that operate on short data sets.

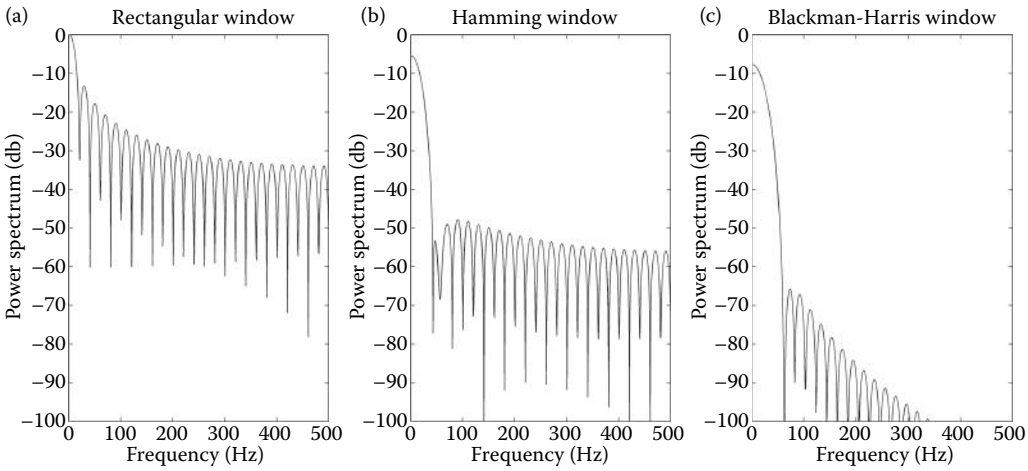

Figure 3.13 (a) The spectrum of a rectangular window. (b) The spectrum of a Hamming window. (c) The spectrum of the Blackman–Harris window.

The Hamming window has the shape of a raised half-sine wave (Figure 3.14) and when the truncated signal is multiplied by this function, the endpoint discontinuities are reduced. The equation for the Hamming function is

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \tag{3.26}$$

where $w[n]$ is the Hamming window function and where $-N/2 \leq n < N/2$ which makes the window length N, the length of the stored signal. As shown in Figure 3.13b, the spectrum of

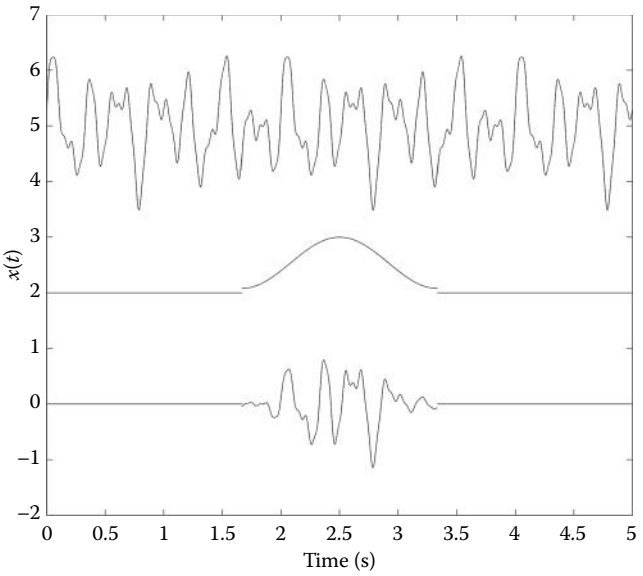

Figure 3.14 The Hamming window, shown as a time function, $w(t)$, is used to truncate a signal, $x(t)$. The Hamming window is defined in Equation 3.26.

the Hamming window has lower sidelobes than that of a rectangular window, which means the nearby peaks have less of an influence on a small peak. However, the mainlobe is wider than that of the rectangular window, leading to a decrease in spectral resolution. When the task is to precisely identify the frequency of a spectral peak, a rectangular window is better.

Many other windows exist and most are structured to reduce the sidelobes, but at the cost of a wider mainlobe. Figure 3.13c shows the spectrum of the Blackman–Harris window, one of the many windows offered in MATLAB's Signal Processing Toolbox. The spectrum of this window has the lowest sidelobes (around –70 dB, Figure 3.13c), but it has a wider mainlobe than the other two windows. The Blackman–Harris window is based on multiple cosines:

$$w[n] = a_0 + a_1\cos\left(\frac{2\pi}{N}n\right) + a_2\cos\left(\frac{2\pi}{N}2n\right) + a_3\cos\left(\frac{2\pi}{N}3n\right) \tag{3.27}$$

where $-N/2 \le n < N/2$ to make the window length N. Typical coefficients are

$$a_0 = 0.35875,\ a_1 = 0.48829,\ a_2 = 0.14128, \text{ and } a_3 = 0.01168$$

The Blackman–Harris window suppresses the influence of nearby frequencies, but at an even greater reduction in spectral resolution. The fact that the Hamming window represents a good compromise in resolution and suppression may explain its adoption as the default window in many MATLAB routines.

Many different windows are easily implemented in MATLAB, especially with the Signal Processing Toolbox. Selecting the appropriate window, like so many other aspects of signal analysis, depends on what spectral features are of interest. If the task is to precisely define the frequency where a sharp spectral peak occurs, then a window with the narrowest mainlobe (the rectangular window) is preferred. If the task is to separate a strong and a weak sinusoidal signal at closely spaced frequencies, then a window with rapidly decaying sidelobes is preferred. Often, the most appropriate window is selected by trial and error.

MATLAB has a number of data windows available. These include the Hamming (Equation 3.26) and the Blackman–Harris (Equation 3.27) windows. The relevant MATLAB routines generate an N-point vector array containing the appropriate window shape. There are several ways of defining a window function in MATLAB along with a special tool, `wintool` for evaluating the various window functions. The most straightforward way to define a window is to call the MATLAB routine that has the same name as the window function.

```
w = window_name(N);      % Generate vector w of length N
                         % containing the function.
```

where `N` is the number of points in the output vector and `window _ name` is the name, or an abbreviation of the name, of the desired window. At this writing, 16 different windows are available in MATLAB's Signal Processing Toolbox. Using `help window` will provide a list of window names. A few of the more popular windows are `bartlett, blackman-harris, gausswin, hamming, hanning,`* and `kaiser`. A few of the routines have additional optional arguments. For example, `chebwin` (the Chebyshev window), which features a nondecaying, constant level of sidelobes, has a second argument to specify the sidelobe amplitude. Of course, the smaller this level is set, the wider the mainlobe becomes and the poorer the frequency resolution. The details for any given window can be found through

* The `hanning` routine implements a Hann window that is very similar to the Hamming window: it is a pure half-cosine without the small offset in the Hamming window (see Figure 3.14, center trace). The similarity in the mathematical function and the name can be a source of confusion.

the help command. In addition to the individual functions, all the window functions can be constructed with one call:

```
w = window(@name,N,opt);      % Construct N-point window 'name.'
```

where name is the name of the specific window function (preceded by "@"), N is the number of points desired, and opt is the possible optional argument(s) required by some specific windows.

To apply a window, simply multiply, point by point, the digitized waveform by the window vector. Of course, the waveform and the window must be the same length. This operation is illustrated in the next example.

EXAMPLE 3.5

Generate a data set consisting of two sine waves closely spaced in frequency (235 and 250 Hz) with added white noise in a 128-point array sampled at 1 kHz. The SNR should be –3 dB. Apply a rectangular, Hamming, and Blackman–Harris window to the signal and take the FT. Plot the magnitude spectra (only the relevant points).

Solution

After defining f_s and N, generate the data using sig_noise. Also, generate a frequency vector for plotting the spectra. Take the magnitude of the FFT output for the signal with no window as the rectangular window. Then multiply the signal by a Hamming window generated by MATLAB's hamming routine. Note that the multiplication should be point by point using MATLAB's.* operator and the window function, and the signal may need to be transposed to have the same vector orientation. Repeat the analysis using the Blackman–Harris window.

```
% Example 3.5 Application of several window functions
%
fs = 1000;                          % Sampling freq assumed by sig_noise
N = 128;
x = sig_noise([235 250],-3,N);      % Generate data
f = (0:N-1)*fs/N;                   % Frequency vector
%
X_mag = abs(fft(x));                % Mag. spect: rect. (no) window
subplot(3,1,1);
plot(f(1:N/2),X_mag(1:N/2));        % Plot magnitude
......labels.............
%
x1 = x .*hamming(N)';               % Apply Hamming window (Eq. 3.26)
X_mag = abs(fft(x1));               % Mag. spect: Hamming window
subplot(3,1,2);
plot(f(1:N/2),X_mag(1:N/2),'k');    % Plot magnitude
.......labels.............
%
x1 = x .*blackmanharris(N)';        % Apply Blackman-Harris (Eq. 3.27)
X_mag = abs(fft(x1));               % Mag. spect: Blackman-Harris window
subplot(3,1,3);
plot(f(1:N/2),X_mag(1:N/2),'k');    % Plot magnitude
......labels.............
```

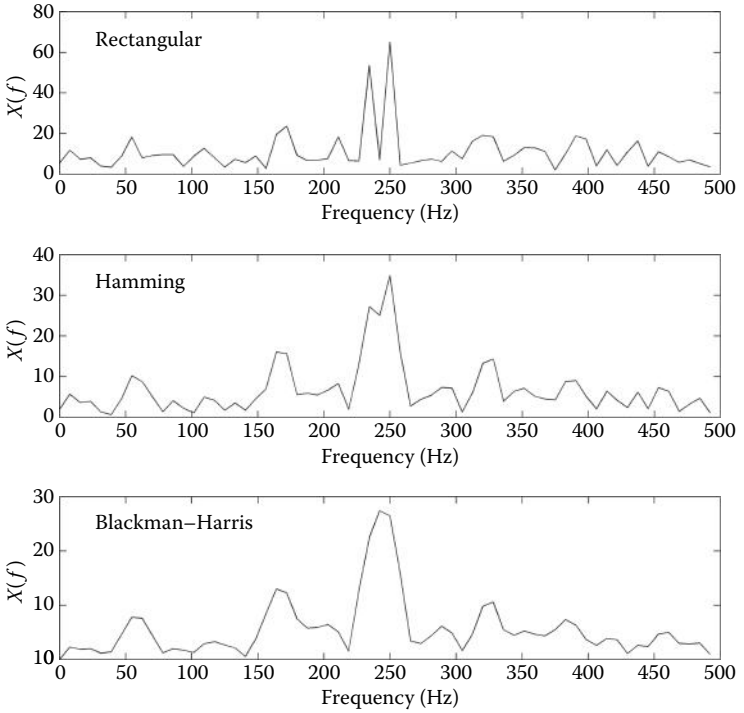

Figure 3.15 Three spectra computed from the same data but using different window functions. The data array is 128-points long and consists of two closely spaced sinusoids (235 and 250 Hz) in noise (SNR –3 dB). The influence of the three windows, with their increasing mainlobe widths, can be seen as a merging of nearby frequencies.

Results

Figure 3.15 shows three spectra obtained from Example 3.5. The differences in the three spectra are due to the windowing. The two peaks are clearly seen in the upper spectrum obtained using a rectangular function; however, the two peaks are barely discernible in the middle spectrum that used the Hamming window due to the broader mainlobe of this window that effectively averages adjacent frequencies. The effect of this frequency merging is even stronger when the Blackman–Harris window is used as only a single broad peak can be seen. The Blackman–Harris filter does provide a somewhat better estimate of the background white noise, as this background spectrum is smoother than in the other two spectra. The smoothing or averaging of adjacent frequencies by the Blackman–Harris window may seem like a disadvantage in this case, but with real data, where the spectrum may be highly variable, this averaging quality can be of benefit.

If the data set is fairly long (perhaps 256 points or more), the benefits of a nonrectangular window are slight. Figure 3.16 shows the spectra obtained with a rectangular and Hamming window to be nearly the same except for a scale difference produced by the Hamming window.

3.3 Power Spectrum

The power spectrum (PS) is commonly defined as the FT of the autocorrelation function. In continuous and discrete notations, the PS equation becomes

$$PS(f) = \frac{1}{T}\int_0^T r_{xx}(t)e^{-j2\pi mf_1 t}dt \quad m = 0,1,2,3,\ldots \tag{3.28}$$

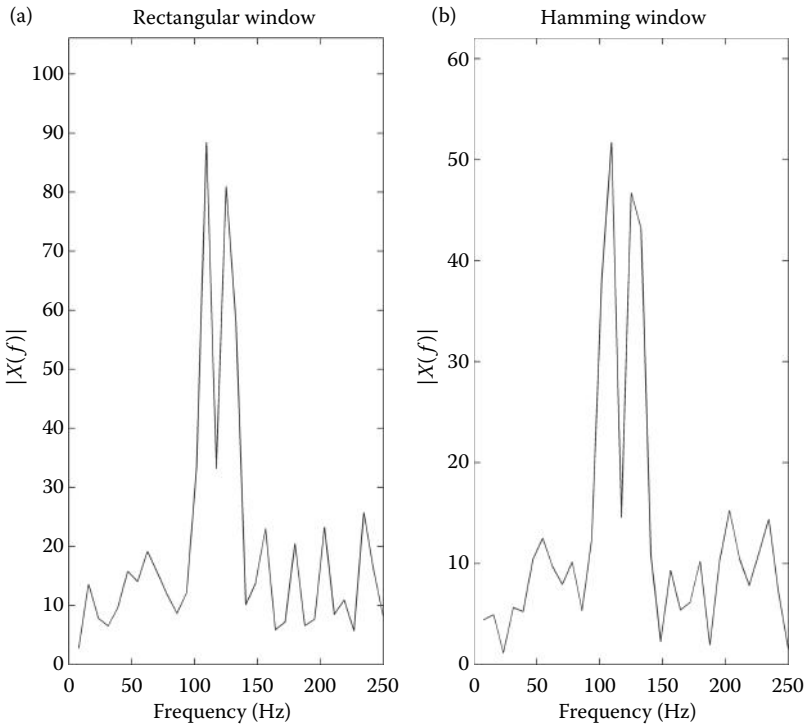

Figure 3.16 The spectra obtained from a signal containing two sinusoids and noise. The signal length is $N = 512$. The use of a nonrectangular window, in this case, a Hamming window, has little effect on the resulting spectrum. In a case where the signal consists of a fairly large number of points (>256), the rectangular window is preferred. Finally, the application of a nonrectangular window tapers the signal and reduces the energy in the signal. When this is of consequence, the scale of the magnitude spectrum can be restored by scaling the signal (or magnitude spectrum) by the ratio of the area under a rectangular window to the area under the applied window. An example of this is given in Problems 3.20 and 3.21, and in Example 3.7.

$$PS[m] = \sum_{n=1}^{N} r_{xx}[n] e^{-\frac{j2\pi mn}{N}} \quad m = 0,1,2,3,\ldots,N/2 \tag{3.29}$$

where $r_{xx}(t)$ and $r_{xx}[n]$ are autocorrelation functions as described in Chapter 2. Since the autocorrelation function has even symmetry, the sine terms of the Fourier series are all zero (see Table 3.1) and the two equations can be simplified to include only real cosine terms:

$$PS[m] = \sum_{n=0}^{N-1} r_{xx}[n] \cos^{\left(2\pi nm/N\right)} \quad m = 0,1,2,3,\ldots,N/2 \tag{3.30}$$

$$PS(f) = \frac{1}{T}\int_{0}^{T} r_{xx}(t)\ \cos(2\pi mft)dt \quad m = 0,1,2,3,\ldots \tag{3.31}$$

Equations 3.30 and 3.31 are sometimes referred to as *cosine transforms*.

A more popular method for evaluating the PS is the *direct approach*. The direct approach is motivated by the fact that the energy contained in an analog signal, $x(t)$, is related to the magnitude of the signal squared integrated over time:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 \, dt \tag{3.32}$$

By an extension of a theorem attributed to Parseval, it can be shown that

$$\int_{-\infty}^{\infty} |x(t)|^2 \, dt = \int_{-\infty}^{\infty} |x(f)|^2 \, df \tag{3.33}$$

Hence, $|X(f)|^2$ equals the energy density function over frequency, also referred to as the *energy spectral density*, *power spectral density* (PSD) or, simply and most commonly, the *power spectrum* (PS). In the direct approach, the PS is calculated as the magnitude squared of the Fourier transform (or the Fourier series) of the waveform of interest:

$$PS(f) = |X(f)|^2 \tag{3.34}$$

This direct approach of Equation 3.34 has displaced the cosine transform for determining the PS because of the efficiency of the FFT. (However, a variation of this approach is still used in some advanced signal-processing techniques involving time and frequency transformation.) Problem 3.18 compares the PS obtained using the direct approach of Equation 3.34 with the traditional cosine transform method represented by Equation 3.30 and, if done correctly, shows them to be identical.

Unlike the FT, the PS does not contain phase information; so, the PS is not an invertible transformation: it is not possible to reconstruct the signal from the PS. However, the PS has a wider range of applicability and can be defined for some signals that do not have a meaningful FT (such as those resulting from random processes). Since the PC does not contain phase information, it is applied in situations where phase is not considered useful or to data that contain a lot of noise, as phase information is easily corrupted by noise.

An example of the descriptive properties of the PS is given using the heart rate data shown in Figure 2.18. The heart rates show major differences in the mean and standard deviation of the rate between meditative and normal states. Applying the autocovariance to the meditative heart rate data (Example 2.11) indicates a possible repetitive structure for the variation in heart rate during meditation. The next example uses the PS to search for structure in the frequency characteristics of both normal and meditative heart rate data.

EXAMPLE 3.6

Determine and plot the power spectra of heart rate variability during both normal and meditative states.

Solution

The PS can be obtained through the Fourier transform using the direct method given in Equation 3.34. However, the heart rate data should first be converted into evenly sampled time data and this is a bit tricky. The data set obtained by downloading from the PhysioNet database provides the heart rate at unevenly spaced times, where the sample times are provided as a second vector. These interval data need to be rearranged into evenly spaced time positions. This process, known as *resampling*, is done through interpolation using MATLAB's `interp1` routine. This routine takes in the unevenly spaced x–y pairs as two vectors along with a vector containing the desired evenly spaced x values. The routine then uses linear interpolation (other options are possible) to approximate the y values that match the evenly spaced x values. The details can be found in the MATLAB help file for `interp1`.

In the program below, the uneven *x*–*y* pairs for the normal conditions are in vectors `t_pre` and `hr_pre`, respectively, both in the MATLAB file `Hr_pre`. (For meditative conditions, the vectors are named `t_med` and `hr_med` and are in file `Hr_med`). The evenly spaced time vector is `xi` and the resampled heart rate values are in vector `yi`.

```
% Example 3.6
% Frequency analysis of heart rate data in the normal and meditative state
%
fs = 100;              % Sample frequency (100 Hz)
ts = 1/fs;             % Sample interval
load Hr_pre;           % Load normal and meditative data
%
% Convert to evenly-spaced time data using interpolation; i.e., resampling
% First generate evenly space time vectors having one second
% intervals and extending over the time range of the data
%
xi = (ceil(t_pre(1)):ts:floor(t_pre(end)));  % Evenly-spaced time vector
yi = interp1(t_pre,hr_pre,xi");              % Interpolate
yi = diff(yi);                               % Remove average
N2 = round(length(yi)/2);
f = (1:N2)*fs/N2);                          % Vector for plotting
%
% Now determine the Power spectrum
YI = abs((fft(yi)).^2);                     % Direct approach (Eq. 3.34)
subplot(1,2,1);
   plot(f,YI(2:N2 + 1,"k");                 % Plot spectrum, but not DC value
   axis([0 .15 0 max(YI)*1.25]);            % Limit frequency axis to 0.15 Hz
 .........label and axis..........
%
% Repeat for meditative data
```

Analysis

To convert the heart rate data into a sequence of evenly spaced points in time, a time vector, `xi`, is first created that increases in increments of 0.01 s ($1/f_s = 1/100$) between the lowest and highest values of time (rounded appropriately) in the original data. A 100-Hz resampling frequency is used here because this is common in heart rate variability studies that use certain nonlinear methods, but in this example, a wide range of resampling frequencies gives the same result. Evenly spaced time data, `yi`, were generated using the MATLAB interpolation routine `interp1`. This example asked for the PS of heart rate *variability*, not heart rate per se; in other words, the change in beat-to-beat rate. To get this beat-to-beat change, we need to take the difference between sample values using MATLAB's `diff` operator before evaluating the PS.

After interpolation and removal of the mean heart rate, the PS is determined using `fft` and then taking the square of the magnitude component. The frequency plots (Figure 3.17) are limited to a maximum of 0.15 Hz since this is where most of the spectral energy is to be found. Note that the DC term is not plotted.

Results

The PS of normal heart rate variability is low and increases slightly with frequency (Figure 3.17a). The meditative spectrum (Figure 3.17b) shows a large peak at around 0.12 Hz, indicating that some resonant process is active at these frequencies, which correspond to a time frame of around 8 s. Speculation as to the mechanism behind this heart rate rhythm is left to the reader.

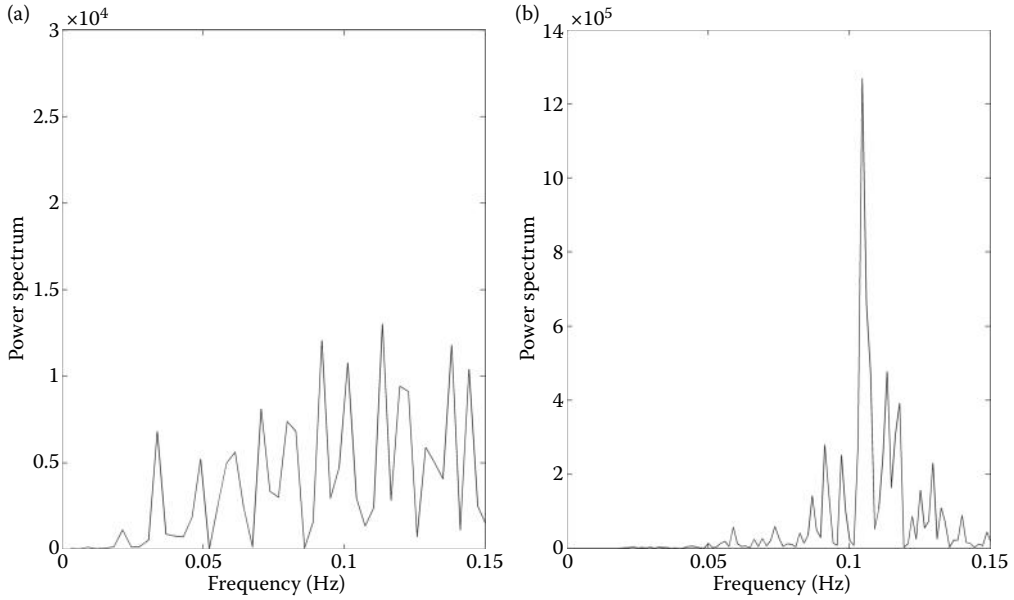

Figure 3.17 (a) The PS of heart rate variability under normal conditions. The power increases slightly with frequency. (b) The PS of heart rate variability during meditation. Strong peaks in power occur around 0.12 Hz, indicating that an oscillatory or resonant process is active around this frequency (possibly a feedback process with a time scale of 8.0 s). Note the much larger amplitude scale of this meditative spectrum.

3.4 Spectral Averaging: Welch's Method

While the PS is usually calculated using the entire waveform, it can also be applied to isolated segments of the data. The power spectra determined from each of these segments can then be averaged to produce a spectrum that represents the broadband or "global," features of the spectrum better. This approach is popular when the available waveform is only a sample of a longer signal. In such situations, spectral analysis is necessarily an estimation process and averaging improves the statistical properties of the result. When the PS is based on a direct application of the FT followed by averaging, it is referred to as an *average periodogram*.

Averaging is usually achieved by dividing the waveform into a number of segments, possibly overlapping, and evaluating the PS on each of these segments (Figure 3.18). The final spectrum is constructed from the ensemble average of the power spectra obtained from each segment. The ensemble average is an averaged spectrum obtained by taking the average over all spectra at each frequency.* Ensemble averaging can be easily implemented in MATLAB by placing the spectra to be averaged in a matrix where each PS is a row of the matrix. The MATLAB averaging routine `mean` produces an average of each column; so, if the spectra are arranged as rows in the matrix, the routine will produce the ensemble average.

This averaging approach can only be applied to the magnitude spectrum or PS because it is insensitive to time translation. Applying this averaging technique to the standard FT would not make sense because the phase spectrum is sensitive to the segment position. Averaging phases obtained from different time positions would be meaningless.

One of the most popular procedures to evaluate the average periodogram is attributed to Welch and is a modification of the segmentation scheme originally developed by Bartlett. In

* Ensemble averaging is also used in the time domain to reduce noise. This application of averaging is described in Chapter 4.

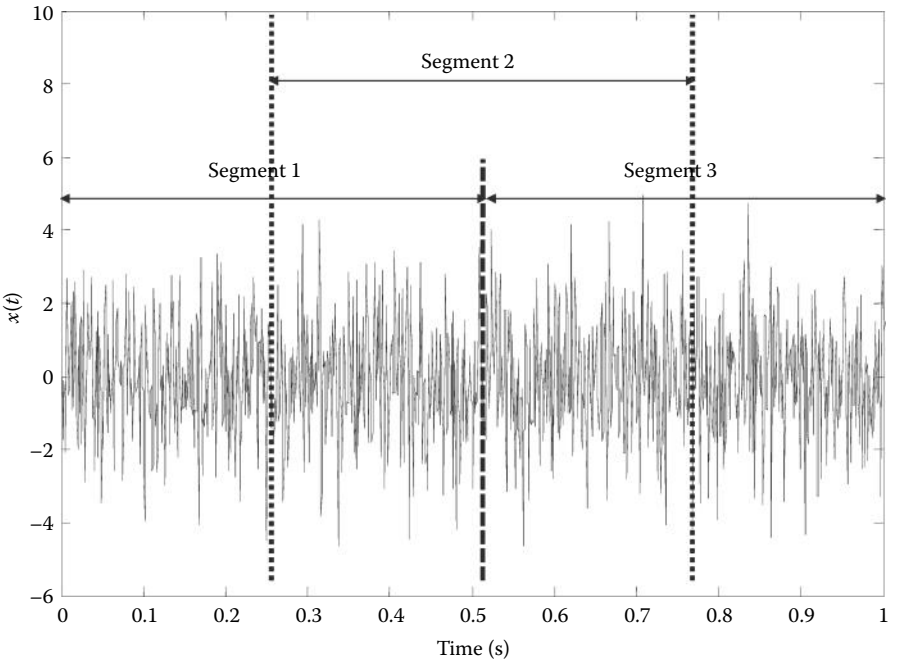

Figure 3.18 A waveform is divided into three segments with a 50% overlap. In the *Welch* method of spectral analysis, the PS of each segment is taken and an average of the three spectra is computed.

this approach, overlapping segments are used and a shaping window (i.e., a nonrectangular window) is sometimes applied to each segment. *Averaged periodograms* traditionally average spectra from half-overlapping segments, that is, segments that overlap by 50% as in Figure 3.18. Higher amounts of overlap have been recommended in applications when computing time is not a factor. Maximum overlap occurs when each segment is shifted by only one sample.

Segmenting the data reduces the number of data samples analyzed to the number in each segment. Equation 3.6 states that frequency resolution is proportional to f_s/N where N is now the number of samples in a segment. So, averaging produces a trade-off between spectral resolution, which is reduced by averaging and statistical reliability. Choosing a short segment length (a small N) will provide more segments for averaging and improve the reliability of the spectral estimate, but will also decrease frequency resolution. This trade-off is explored in the next example.

While it is not difficult to write a program to segment the waveform and average the individual power spectra, it is unnecessary as MATLAB has a routine that does this. The Signal Processing Toolbox includes the function `pwelch`* that performs these operations:

```
[PS,f] = pwelch(x,window,noverlap,nfft,fs); % Apply the Welch method
```

Only the first input argument, `x`, the data vector is required as the other arguments have default values. By default, `x` is divided into eight sections with 50% overlap, each section is

* The calling structure for this function is different in MATLAB versions <6.1. Use the help file to determine the calling structure if you are using an older version of MATLAB. Another version, `welch`, is included in the routines associated with this chapter and can be used if the Signal Processing Toolbox is not available. This version always defaults to the Hamming window but otherwise offers the same options as `pwelch`. See the associated help file.

windowed with the default Hamming window, and eight periodograms are computed and averaged. If `window` is an integer, it specifies the segment length and a Hamming window of that length is applied to each segment. If `window` is a vector, then it is assumed to contain the window function itself. Many window functions are easily implemented using the window routines described below. The segment length analyzed can be shortened by making `nfft` less than the window length (`nfft` must be less than, or equal to window length). The argument `noverlap` specifies the overlap in samples. The sampling frequency is specified by the optional argument `fs` and is used to fill the frequency vector, `f`, in the output with appropriate values. As is always the case in MATLAB, any variable can be omitted and the default can be selected by entering an empty vector, [ ]. The output PS is in vector `PS` and is only half the length of the data vector, x, as the redundant points have been removed. Check the help file for other options. The spectral modification produced by the Welch method is explored in the following example.

EXAMPLE 3.7

Evaluate the influence of averaging power spectra on a signal made up of a combination of broadband and narrowband signals along with added noise. The data can be found in file `broadband1.mat`, which contains a white-noise signal filtered to be between 0 and 300 Hz and two closely spaced sinusoids at 390 and 410 Hz.

Solution

Load the test data file `broadband1` containing the narrowband and broadband processes. First, calculate and display the unaveraged PS using Equation 3.34. Then apply PS averaging using an averaging routine. Use a segment length of 128 points with a maximum overlap of 127 points. To implement averaging, use `pwelch` with the appropriate calling parameters.

```
% Example 3.7 Investigation of the use of averaging to improve
% broadband spectral characteristics in the power spectrum.
%
load broadband1;                % Load data (variable x)
fs = 1000;                      % Sampling frequency
nfft = 128;                     % Segment length
%
% Un-averaged spectrum using the direct method of Eq. 3.34
%
PS = abs((fft(x)).^2)/length(x);          % Calculate un-averaged PS
half_length = fix(length(PS)/2);          % Valid points
f = (0:half_length-1)*fs/(2*half_length);     % Frequency vector for plotting
subplot(1,2,1)
plot(f,PS(1:half_length),'k');            % Plot un-averaged Power Spectrum
     .........lables and title........
%
 [PS_avg,f] = pwelch(x,nfft, nfft-1,[], fs);  % Periodogram, max. overlap
%
subplot(1,2,2)
    plot(f,PS_avg,'k');                   % Plot periodogram
    .......labels and title.......
```

Analysis

This example uses `pwelch` to determine the averaged PS and also calculates the unaveraged spectrum using `fft`. Note that in the latter case, the frequency vector includes the DC term. For

the averaged spectrum or the periodogram, a segment length of 128 (a power of 2) was chosen along with maximal overlap. In practice, the selection of segment length and the averaging strategy is usually based on experimentation with the data.

Results

In the unaveraged PS (Figure 3.19a), the two sinusoids at 390 and 410 Hz are clearly seen; however, the broadband signal is noisy and poorly defined. The periodogram produced from the segmented and averaged data in Figure 3.19b is much smoother, reflecting the constant energy in white noise better, but the loss in frequency resolution is apparent as the two high-frequency sinusoids are hardly visible. This demonstrates one of those all-so-common engineering compromises. Spectral techniques that produce a good representation of "global" features such as broadband features are not good at resolving narrowband or "local" features such as sinusoids and vice versa.

EXAMPLE 3.8

Find the approximate bandwidth of the broadband signal, `x`, in file `Ex3_8_data.mat` (f_s = 500 Hz). The unaveraged PS of this signal is shown in Figure 3.20.

Solution

The example presents a number of challenges similar to those found in real-world problems. First, the PS obtained using the direct method is very noisy, as is often the case for broadband signals (Figure 3.20). To smooth this spectrum, we can use spectral averaging; this produces

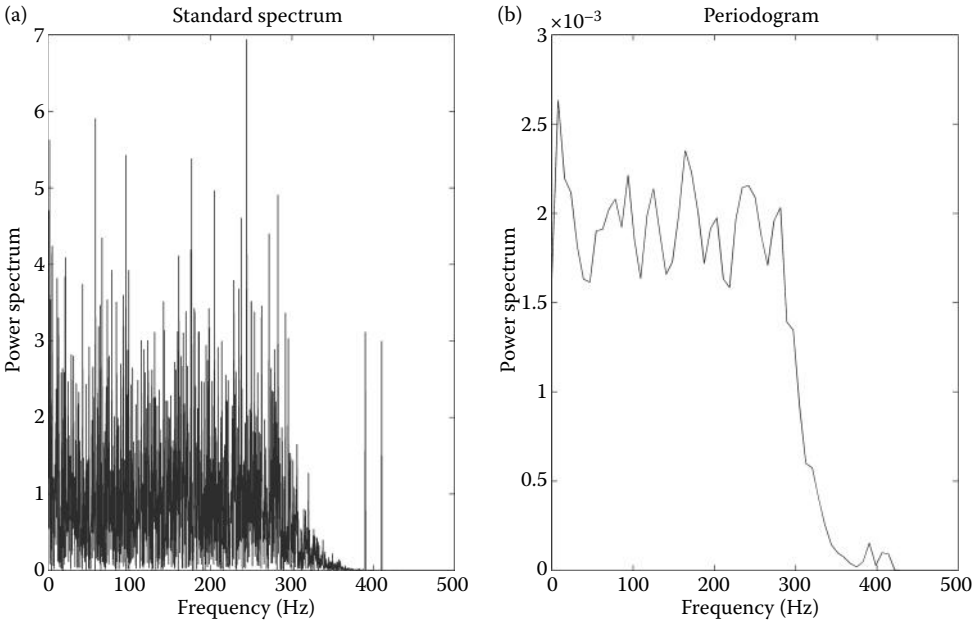

Figure 3.19 A waveform consisting of a broadband signal and two high-frequency sinusoids with noise. (a) The unaveraged PS clearly shows the high-frequency peaks, but the broadband characteristic is unclear. (b) The Welch averaging technique describes the spectrally flat broadband signal better, but the two sinusoids around 400 Hz are barely visible and would not be noticed unless you already knew they were there.

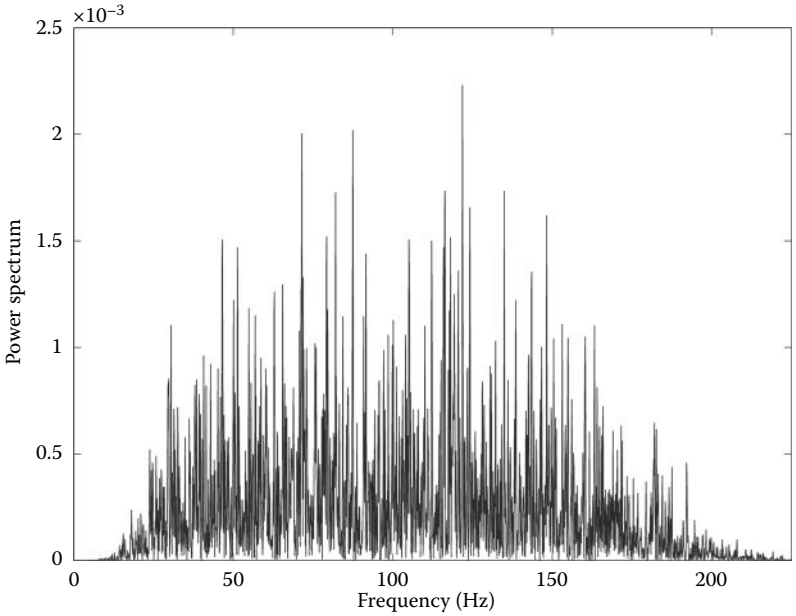

Figure 3.20 The PS of the broadband signal used in Example 3.8. Spectral averaging is used to produce a smoother spectrum (see Figure 3.21), making it easier to identify the cutoff frequencies. Since this is a PS, the cutoff frequencies will be taken at 0.5 (0.707^2), the bandpass value. This is the same as the –3-dB point in the magnitude spectrum when plotted in dB.

the cleaner spectrum shown in Figure 3.21. However, to find the bandwidth requires an estimate of spectral values in the bandpass region and there is considerable variation in this region.

To estimate the bandpass values, we take the average of all spectral values within 50% of the maximum value. This threshold is somewhat arbitrary, but seems to work well. The bandpass value obtained using this strategy is plotted as a dashed horizontal line in Figure 3.21. To find the high and low cutoff frequencies, we can use MATLAB's `find` routine to search for the first and last points >0.5 of the bandpass value. If this was the magnitude spectrum, we would take 0.707 of the bandpass value as described in Chapter 1, but since the PS is the square of the magnitude spectrum (Equation 3.34), we use 0.5 (i.e., 0.707^2) as the cutoff frequency value.

```
% Example 3.8 Find bandwidth of the signal x in file Ex3_8_data.mat
%
load Ex3_8_data.mat;           % Load the data file. Data in x.
fs = 500;                      % Sampling frequency (given).
nfft =64;                      % Power spectrum window size
[PS,f] = pwelch(x,[],nfft-1,nfft,fs);       % PS using max. overlap
plot(f,PS,'r'); hold on;                    % Plot spectrum
bandpass_thresh = 0.50 *max(PS);            % Threshold for bandpass values
bandpass_value = mean(PS(PS > bandpass_thresh));    % Use logical indexing
plot(f,PS,'k'); hold on;                      % Plot spectrum
plot([f(1) f(end)], [bandpass_value bandpass_value],':k');  % Bandpass est.
      .......labels.........
%
cutoff_thresh = 0.5 *bandpass_value;          % Set cutoff threshold
in_fl = find(PS >cutoff_thresh, 1, 'first');  % Find index of low freq. cutoff
```

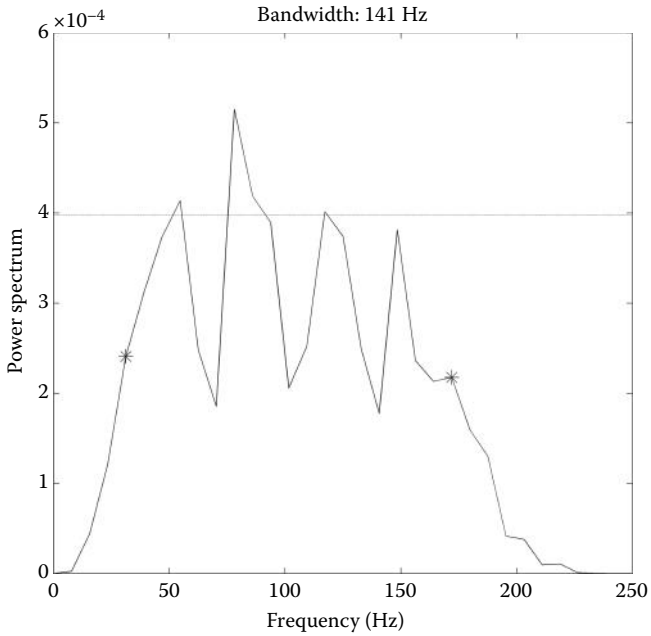

Figure 3.21 The spectrum of the signal used in Example 3.8 obtained using spectral averaging. The window size was selected empirically to be 64 samples. This size appears to give a smooth spectrum while maintaining good resolution. The bandpass value is determined by taking the average of spectral values greater than half the maximum value. The cutoff frequency points (these are the same as the –3-dB point in the magnitude spectrum) are determined by searching for the first and last points in the spectrum that are >0.5 of the average bandpass value. MATLAB's `find` routine was used to locate these points that are identified as large dots on the spectral curve. The frequency vector is used to convert the index of these points into equivalent frequencies and the calculated bandwidth is shown in the figure title.

```
in_fh = find(PS >cutoff_thresh, 1, 'last');   % Find index of high freq. cutoff
f_low = f(in_fl);                             % Find low cutoff freq.
f_high = f(in_fh);                            % Find high cutoff freq.
plot(f_low,PS(in_fl),'k*','MarkerSize',10);  % Put marker at cutoff freqs.
plot(f_high,PS(in_fh),'k*','MarkerSize',10);
BW = f_high - f_low;                          % Calculate bandwidth
title(["Bandwidth: ',num2str(BW,3), 'Hz'],'FontSize',14);
```

Results

Using the `pwelch` routine with a window size of 64 samples produces a spectrum that is fairly smooth, but still has a reasonable spectral resolution (Figure 3.21). As is so often necessary in signal processing, this window size was found by trial and error. Using *logical indexing* (`PS(PS > bandpass_thresh)`) allows us to get the mean of all spectral values >50% of the maximum power spectral value. This value is plotted as a horizontal line on the spectral plot.

Using MATLAB's `find` routine with the proper options gives us the indices of the first and last points above 0.5 of the bandpass value ($f_{low} = 31$ Hz and $f_{high} = 172$ Hz). These are plotted and superimposed on the spectral plot (large dots in Figure 3.21). The high and low sample points can be converted into frequencies using the frequency vector produced by `pwelch`. The difference between the two cutoff frequencies is the bandwidth and is displayed

in the title of the spectral plot (Figure 3.21). The estimation of the high and low cutoff frequencies might be improved by interpolating between the spectral frequencies on either side of the 0.5 values. This is done in Problem 3.36.

3.5 Summary

The sinusoid (i.e., $A \cos(\omega t + \theta)$) is a pure signal in that it has energy at only one frequency, the only waveform to have this property. This means that sinusoids can serve as intermediaries between the time-domain representation of a signal and its frequency-domain representation. The technique for determining the sinusoidal series representation of a periodic signal is known as Fourier series analysis. To determine the Fourier series, the signal of interest is correlated with sinusoids at harmonically related frequencies. This correlation provides either the amplitude of a cosine and sine series or the amplitude and phase of a sinusoidal series. The latter can be plotted against frequency to describe the frequency-domain composition of the signal. Fourier series analysis is often described and implemented using the complex representation of a sinusoid. A high-speed algorithm exists for calculating the discrete time Fourier series, also known as the DFT. The FT is invertable and the inverse FT can be used to construct a time-domain signal from its frequency representation.

Signals are usually truncated or shortened to fit within the computer memory. If the signal is simply cut off at the beginning and end, it is the same as multiplying the original signal by a rectangular window of amplitude 1.0 having the length of the stored version. Alternatively, the digital version of the signal can be multiplied by a shaping window that tapers the signal ends, providing less-abrupt endpoints. The window shape will have an influence on the resulting signal spectrum, particularly if the signal consists of a small number of data points. Window selection is a matter of trial and error, but the Hamming window is popular and is the default in some MATLAB routines that deal with short data segments. Window functions are used in the construction of some filters, as is described in the next chapter.

The DFT can be used to construct the PS of a signal. The power spectral curve describes how the signal's power varies with frequency. The PS is particularly useful for random data where phase characteristics have little meaning. By dividing the signal into a number of possibly overlapping segments and averaging the spectrum obtained from each segment, a smoothed PS can be obtained. The resulting frequency curve will emphasize the broadband or global characteristics of a signal's spectrum, but will lose the fine detail or local characteristics.

PROBLEMS

3.1 Extend Example 3.1 to include $m = N$ frequencies, where N is the number of points in the signal. Since $Mf_1 = f_s$, all the magnitude frequency components above $N/2$ should be reflections of frequency components below $N/2$. Note that the phase characteristics also reflect on either side of $f_s/2$ but are also inverted.

3.2 Modify Example 3.2 to decompose the waveform into 100 components and also the maximum number of valid components (i.e., $N/2$), and recompose it using both 100 and $N/2$ components. Plot only the reconstructed waveforms to best show the differences in reconstruction. Note the Gibbs oscillations with less than the maximum number of reconstructions.

3.3 Modify Example 3.2 to use the sawtooth waveform given by the equation:

$$x(t) = \begin{cases} t & 0 < t \leq 0.5 \\ 0.5 - t & 0.5 < t \leq 1.0 \end{cases}$$

Also, reconstruct it using Equation 3.13 as in Example 3.2 with three and six components. Assume a sample interval of 250 Hz. Note how close the reconstruction is, even with only three components. Also note the absence of the Gibbs oscillations because there is no discontinuity in this waveform.

3.4 Modify Example 3.3 to plot all the points generated by the FFT algorithm. Use a signal that consists of two sinusoids at 100 and 200 Hz, both with an SNR of −10 dB. Use `sin _ noise` to construct this signal. Observe that the magnitude plot is a mirror image about $f_s/2$ and the phase plot is the mirror image inverted.

3.5 Construct the sawtooth wave shown below and use the MATLAB `fft` routine to find the frequency spectrum of this waveform. Then reconstruct the square wave using the first 24 components using Equation 3.15. Do not plot the spectrum, but plot the original and reconstructed waveform superimposed. Make $f_s = 1024$ and $N = 1024$ to represent the 1-s periodic waveform. [Hint: The MATLAB `fft` routine does no scaling; so, you will have to scale the output to get the correct reconstructed waveform. Also, remember that the first element of the complex vector produced by the `fft` routine is the DC component (that will be 0.0 for this waveform) and the first complex sinusoidal coefficient is in the *second* element.]

3.6 File `sawtooth.mat` contains vector `x`. This signal is the same waveform as in Problem 3.5 but with an added DC term. For this signal, $f_s = 1024$ and $N = 1024$. As in Problem 3.5, use the `fft` routine to find the frequency spectrum of this waveform. Then reconstruct the signal using the first 24 components and appropriately add the DC term. Do not plot the spectrum, but plot the original and reconstructed waveform superimposed. [Hint: After reconstructing the waveform, add the first `fft` term that contains the DC term, but remember to divide by 2 as indicated in Equation 3.15.]

3.7 Construct the waveform given by the equation below. Make $f_s = 1024$ and $N = 1024$.

$$x(t) = \begin{cases} t & 0 < t \leq 0.5 \\ -t - 0.5 & 0.5 < t \leq 1.0 \end{cases}$$

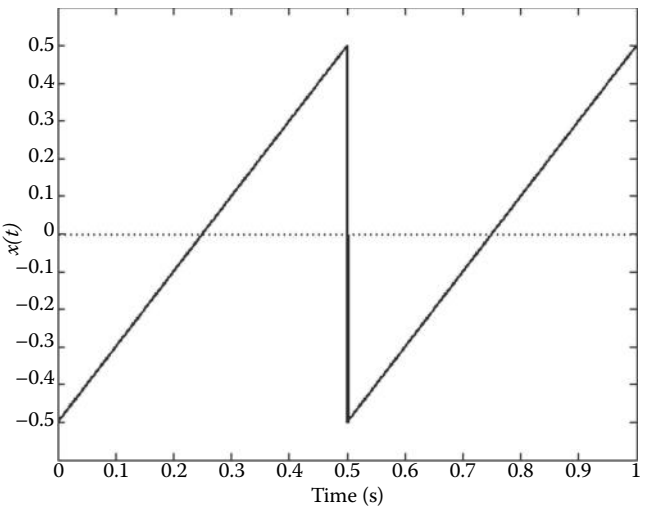

Figure P3.5 Waveform for Problem 3.5.

The second half of the waveform has the inverted version of the first; so, this waveform has half-wave symmetry. Plot the waveform to ensure that you have constructed it correctly. Use `fft` to find the frequency spectrum of this waveform and reconstruct the waveform using the first 24 components as in Problem 3.5 above. Then reconstruct the waveform using Equation 3.15 and only the first 12 odd-numbered harmonics. Remember that since the first element of the `fft` routine's output is the DC component so odd harmonics will be in elements 2, 4, 6, ... Note that the two reconstructions are the same since the even components in this half-wave symmetrical waveform are 0.0.

3.8 Use `randn` to construct two arrays of white noise: one 128 points in length and the other 1024 points in length. Take the FT of both and plot the magnitude of the nonredundant points. Does increasing the length improve the spectral estimate of white noise?

3.9 Use the routine `sig_noise` to generate a waveform containing 200- and 400-Hz sine waves with an SNR of –8 dB, that is, `x = sig_noise([200 400], -8,N)`. Make $N = 512$. Plot the nonredundant magnitude spectrum. Repeat for an SNR of –16 dB. Note that one or the other of the two sinusoids is hard to distinguish at the higher (–16 dB) noise level. Recall that with `sig_noise`, $f_s = 1$ kHz. Rerun the program several times to get a feel for the variability in results due to noise fluctuations.

3.10 Use the routine `sig_noise` to generate a waveform containing 200- and 400-Hz sine waves as in Problem 3.9 with an SNR of –12 dB. Make $N = 1000$ samples ($f_s = 1$ kHz). Again, plot only the nonredundant points in the magnitude spectrum. Repeat for the same SNR, but for a signal with only 200 points. Note that the two sinusoids are hard to distinguish with the smaller data sample. Taken together, Problems 3.9 and 3.10 indicate that both data length and noise level are important when detecting sinusoids (i.e., narrowband signals) in noise. Rerun the program several times to get a feel for the variability in results due to noise fluctuations.

3.11 The data file `pulses.mat` contains three signals: `x1, x2,` and `x3`. These signals are all 1.0 s in length and were sampled at 500 Hz. Plot the three signals and show that each signal contains a single 40-ms pulse, but at three different delays: 0, 100, and 200 ms. Calculate and plot the spectra for the three signals superimposed on a single magnitude and single phase plot. Plot only the first 20 points as discrete points, plus the DC term, using a different color for each signal's spectra. Apply the `unwrap` routine to the phase data and plot in deg. Note that the three magnitude plots are identical and while the phase plots are all straight lines, they have radically different slopes.

3.12 Load the file `chirp.mat` that contains a sinusoidal signal, `x`, which increases its frequency linearly over time. The sampling frequency of this signal is 5000 Hz. This type of signal is called a "chirp" signal because of the sound it makes when played through an audio system. If you have an audio system, you can listen to this signal after loading the file using the MATLAB command: `sound(x,5000);`. Take the FT of this signal and plot magnitude and phase (no DC term). Note that the magnitude spectrum shows the range of frequencies that are present, but there is no information on the timing of these frequencies. Actually, information on signal timing is contained in the phase plot but, as you can see, this plot is not easy to interpret. Advanced signal-processing methods known as "time-frequency" methods (see Chapter 6) are necessary to recover the timing information.

3.13 Load the file `ECG_1 min.mat` that contains 1 min of ECG data in variable `ecg`. Take the FT. Plot both the magnitude and phase (unwrapped) spectrum up to 20 Hz and do not include the DC term. The spectrum will have a number of peaks; however, the largest peak (that is also the lowest in frequency) will correspond to the cardiac cycle. Find the average heart rate in beats/min from this peak. The sample frequency is 250 Hz. [Hint: Get the index of that peak from the second output argument of MATLAB's `max` routine and the actual frequency from the frequency vector value at that index (i.e., `fmax = f(i_max)`). The interbeat interval corresponding to that frequency in seconds. This is just the inverse of that frequency (i.e., `1/fmax`). The heart rate in beats/min is 60 s divided by the interbeat interval.]

3.14 This problem demonstrates aliasing. Generate a 512-point waveform consisting of two sinusoids at 200 and 400 Hz. Assume f_s = 1 kHz. Generate another waveform containing frequencies at 200 and 900 Hz. Take the FT of both waveforms and plot the magnitude of the spectrum up to $f_s/2$. Plot the two spectra superimposed, but plot the second spectrum as dashed and in a different color to highlight the additional peak due to aliasing at 100 Hz. [Hint: To generate the sine waves, first construct a time vector, `t`, then generate the signal using `x = sin(2*pi*f1*t) + sin(2*pi*f2*t)` where `f1` = 200 for both signals, whereas `f2` = 400 for one waveform and `f2` = 900 for the other.]

3.15 Another example of aliasing. Load the chirp signal, `x`, in file `chirp.mat` and plot the magnitude spectrum. Now, decrease the sampling frequency by a factor of 2, removing every other point, and recalculate and replot the magnitude spectrum. Note the distortion of the spectrum produced by aliasing. Do not forget to recalculate the new frequency vector based on the new data length and sampling frequency. (The new sampling frequency is effectively half that of the original signal.) Decreasing the sampling frequency is termed *downsampling* and can be easily done in MATLAB: `x1 = x(1:2:end)`. Note the disturbance in the downsampled signal at the higher frequencies.

3.16 Load the file `sample_rate.mat` that contains signals `x` and `y`. Are either of these signals oversampled (i.e., $f_s/2 \leq f_{max}$)? Alternatively, could the sampling rate of either signal be safely reduced? Justify your answer. Both signals are of the same length and f_s = 500 Hz.

3.17 The file `short.mat` contains a very short signal of 32 samples. Plot the magnitude spectrum as discrete points obtained with and without zero padding. Zero pad out to a total of 256 points. Note the interpolation provided by zero padding.

3.18 Use `sig_noise` to generate a 256-point waveform consisting of a 300-Hz sine wave with an SNR of –12 dB (`x = sig_noise(300,-12,256);`). Calculate and plot the PS using two different approaches. First, use the direct approach: take the FT and square the magnitude function. In the second approach, use the traditional method defined by Equation 3.29: take the FT of the autocorrelation function. Calculate the autocorrelation function using `xcorr`, then take the absolute value of the `fft` of the autocorrelation function. You should only use the second half of the autocorrelation function (those values corresponding to positive lags). Plot the PS derived from both techniques. The scales will be different because of different normalizations.

3.19 Generate a white-noise waveform using `randn`. Make N = 1024 and f_s = 500 Hz. Construct the PS using the direct approach given by Equation 3.34. Apply the Welch methods to the waveform using a Hamming window with a window size (`nfft`) of 128 samples with no overlap. Now, change the overlap to 50% (an overlap

of 64 samples) and note any changes in the spectrum. Submit the three frequency plots as subplots appropriately labeled.

3.20 This and the next problem demonstrate a reduction in energy produced by the use of tapering windows. Use `sig _ noise` to generate a waveform containing two sinusoids, 200 and 300 Hz, with an SNR of –3 dB. Make the waveform fairly short: $N = 64$ ($f_s = 1$ kHz). Take the magnitude spectrum with a rectangular and a Blackman–Harris window and compare. Plot both magnitude spectra (as always, only valid points). Note the lower values for the windowed data. Rescale the time-domain signal to compensate for the reduction in energy caused by the application of the Blackman–Harris window. Replot both spectra and compare. This problem demonstrates that rescaling the time-domain signal when applying a tapering window is essential if magnitude spectra amplitude levels are to be preserved. Note that the spectrum of the rescaled windowed signal now has approximately the same amplitude as the unwindowed signal. Also note the smoothing done by the Blackman–Harris window. [Hint: Use either Equation 3.27 or MATLAB (`w = blackmanharris(N);`) to generate the window. Rescale the windowed time signal by multiplying by a scale factor. The scale factor can be determined by taking the area (i.e., sum) of a rectangular window of amplitude 1.0 divided by the area (sum) under the Blackman–Harris window. The rectangular window can be created using MATLAB's `ones` function.]

3.21 Repeat Problem 3.20 but compare the power spectra instead. Rescale the windowed time-domain signal before taking the windowed PS (again scale by the sum of a rectangular window divided by the sum of the Blackman–Harris window). Use the direct method to calculate both the power spectra (Equation 3.34). Again, compare the rectangular window PS with that obtained from the windowed, unscaled waveform and with the PS obtained from the windowed, scaled waveform.

3.22 Repeat Example 3.6 but determine the PS using the Welch method. Use the default number of segments (8) and the default overlap (50%). Plot the normal and meditative power spectra using the same scale for the vertical axis to compare the energy levels in the two signals better.

3.23 Generate the "filtered" waveform used in Problem 2.36. First, construct a 512-point Gaussian noise array, then filter it by averaging segments of three consecutive samples to produce a new signal. In other words, construct a new vector in which every point is the average of the preceding three points in the noise array: $y[n] = x[n]/3 + x[n–1]/3 + x[n–2]/3$. This is called a *moving average* (MA) filter. You could write the code to do this, but an easier way is to convolve the original data with a function consisting of three equal coefficients having a value of 1/3, in MATLAB: `h(n) = [1/3  1/3  1/3]`. Find the PS of this filtered noise data using the direct method (Equation 3.34). Assume $f_s = 200$ Hz for plotting. Note the lowpass characteristic of this averaging process, as some of the higher frequencies have been removed from the white noise.

3.24 Repeat Problem 3.23 above using the Welch averaging method with 256 and 32 segment lengths. Use the default window and maximum overlap. Use `subplot` to plot the two power spectra on the same page for comparison. The actual frequency characteristics of the MA filter begin to emerge when the short window is used.

3.25 This problem examines signal detection in noise using the Welch averaging method with different segment lengths. Load file `welch_ana.mat` that contains waveform `x` consisting of two sinusoids at 140 and 180 Hz with an SNR of –14 dB. $N = 256$ and $f_s = 1$ kHz. Construct the PS using window sizes of 256 (i.e., N), 128,

64, and 32. Use the default window type and overlap. Note that at the smallest window, the two narrowband signals can no longer be distinguished from one another.

3.26 Construct the first-order process defined by the impulse response equation below.

$$h(t) = e^{(-t/\tau)}$$

To calculate the time vector, use $N = 1000$ and $f_s = 200$ Hz. Use convolution to input white noise to the process and use the Welch method to calculate the PS of the output (i.e., the result of convolution). Use a window size of 128 and the default overlap. Find and plot the PS for two time constants: $\tau = 0.2$ and 2.0 s. Limit the spectral plot to be between 0 and 10 Hz.

3.27 Repeat Problem 3.26, but use a second-order process given by the equation below:

$$h(t) = e^{(-2\pi t)} \sin(20\pi t)$$

Use the same variables as in Problem 3.26 to construct the equation: $N = 1000$ and $f_s = 200$ Hz. Use the Welch method to determine and plot the PS of the process to white-noise input. Use window sizes of 256 and 64. Limit the frequency range of the plot to be between 0 and 30 Hz.

3.28 Use `sig_noise` to generate a signal that contains a combination of 200- and 300-Hz sine waves with SNRs of −4 dB. Make the data segment fairly short, 64 samples, and recall $f_s = 1$ kHz. Plot the magnitude spectrum obtained with a rectangular, a Hamming, and a Blackman–Harris window. Generate 512-sample Hamming and Blackman–Harris windows using Equations 3.26 and 3.27. Multiply these windows point by point with the signal. Plot the magnitude spectra of the three signals. Note that the difference is slight, but could be significant in certain situations.

3.29 Use `sig_noise` to generate a signal that contains a combination of 280- and 300-Hz sine waves with SNRs of −10 dB. Make $N = 512$ samples and recall $f_s = 1$ kHz. Repeat Problem 3.19 and plot the magnitude spectrum obtained with a rectangular, a Hamming, and a Blackman–Harris window. Generate a 512-sample Hamming and Blackman–Harris windows using Equations 3.26 and 3.27. Multiply point by point with the signal. Plot the magnitude spectra of the three signals. Rerun the program several times and observe the variation in spectra that are produced due to variations in the random noise. Note how the Hamming window is a good compromise between the rectangular window (essentially no window) and the Blackman–Harris windows, again motivating its use as the default window in some MATLAB routines.

3.30 Use `sig_noise` to generate two arrays, one 128-points long and the other 512-points long. Include two closely spaced sinusoids having frequencies of 320 and 340 Hz with an SNR of −12 dB. Calculate and plot the (unaveraged) PS using Equation 3.34. Repeat the execution of the program several times and note the variability of the results, indicating that noise is noisy. Recall $f_s = 1$ kHz.

3.31 Use `sig_noise` to generate a 256-point array containing 320- and 340-Hz sinusoids as in Problem 3.21. Calculate and plot the unaveraged PS of this signal for an SNR of −10, −14, and −18 dB. How does the presence of noise affect the ability to detect and distinguish between the two sinusoids?

3.32 Load the file `broadband2` that contains variable `x`, a broadband signal with added noise. Assume a sample frequency of 1 kHz. Calculate the averaged PS using

pwelch. Evaluate the influence of segment length using segment lengths of $N/4$ and $N/16$, where N is the length of the date of variable. Use the default overlap.

3.33 Load the file eeg _ data.mat that contains an ECG signal in variable eeg ($f_s = 50$ Hz). Analyze these data using the unaveraged power spectral technique and an averaging technique using the pwelch routine. Find a segment length that gives a smooth background spectrum, but still retains any important spectral peaks. Use a 99% overlap.

3.34 Load the file broadband3.mat that contains a broadband signal and a sinusoid at 400 Hz in variable x ($f_s = 1$ k Hz). Use the pwelch routine and apply a rectangular and a Blackman–Harris window to the PS. Analyze the spectrum of the combined broadband/narrowband signal with the two windows and with segment lengths of $N/4$ and $N/16$, where N is the length of the date of variable. Use the default (50%) overlap. Use subplot to group the four spectra for easy comparison. Note the trade-off between smoothing of broadband features and preservation of narrowband features. [Hint: Use ones(1,N) to generate a rectangular window of appropriate length.]

3.35 Use the approach given in Example 3.8 to find the approximate bandwidth of the signal x in file broadband2.mat. ($f_s = 500$ Hz). Plot the unaveraged and smoothed spectra. Adjust the window to get the maximum size that still results in a reasonably smooth spectrum. As in Example 3.8, show the bandpass value and the cutoff frequency points.

3.36 Load the file broadband2.mat and find the bandwidth of the signal in x using the methods of Example 3.8 combined with interpolation ($f_s = 1$ kHz). Determine the PS using pwelch with a window size of 64 and maximum overlap. This will give a highly smoothed estimate of the PS but a poor estimate of bandwidth due to the small number of points. Interpolation can be used to improve the bandwidth estimate from the smoothed PS. The easiest way to implement interpolation in this problem is to increase the number of points in the PS using MATLAB's interp routine. (E.g., PS1 = interp(PS,5) would increase the number of points in the PS by a factor of 5.) Estimate the bandwidth using the approach of Example 3.8 before and after expanding the PS by a factor of 5. Be sure to expand the frequency vector (f in Example 3.8) by the same amount. Repeat the steps to find the bandpass values and cutoff frequencies on the expanded spectrum, and plot the bandpass values and cutoff frequencies as in the example. Compare the bandwidth values obtained with and without interpolation.