CHAPTER

4

Signal Analysis in the Frequency Domain—Implications and Applications

4.1 GOALS OF THIS CHAPTER

We are now quite skilled in transforming data into, and out of the frequency domain. We understand how this transformation works by correlating our signal with a series of harmonically related sinusoids. Once we have an equivalent sinusoidal series, it is easy to map the sinusoidal correlation coefficients to magnitude and phase in the frequency domain. In this chapter, we use our expertise in frequency domain signals to explore some important features of signals, particularly the consequences of signal digitization.

To convert a real-world continuous analog signal to discrete data, we know that two major steps are involved: slicing the signal in amplitude and sampling the signal in time. But we do not know how the resulting sliced and diced digitized signal corresponds to the original. Since all of our signal processing tools are applied to discrete signals, it is very important to determine if these signals closely reflect the original analog signals. In addition, it is often necessary to analyze only a portion of the original signal due to computer memory limitations, and we would like to understand how this signal truncation affects our analysis. Frequency domain representation helps us to understand both these limitations.

In signal analysis, usually only the magnitude spectrum of a signal is of interest. This is because the phase spectrum, while necessary to reconstruct the original signal, is difficult to interpret. In such cases, we often use the "power spectrum" which shows the spectral distribution of signal energy. The power spectrum is directly determined from the magnitude spectrum. Although the original signal cannot be reconstructed from the power spectrum, it is very easy to interpret.

To summarize, in this chapter we will:

- Use frequency domain methods to analyze the influence of time sampling on the digitized waveform.
- Define the power spectrum.

Circuits, Signals, and Systems for Bioengineers
https://doi.org/10.1016/B978-0-12-809395-5.00004-7

- Describe the effect of data truncation and introduce the concept of nonrectangular windowing of a signal.
- Define the bandwidth of a signal.
- Show how spectral averaging can be used to emphasize the broadband characteristics of a signal.

4.2 DATA ACQUISITION AND STORAGE

As mentioned, converting a continuous signal to discrete format involves amplitude slicing (quantization) and time sampling. In addition, it may be necessary to convert only a portion of the signal because of memory constraints. We look at each of these operations in turn and determine their influence on the discrete signal stored in memory.

4.2.1 Data Sampling: The Sampling Theorem

Slicing the signal into discrete time intervals (usually evenly spaced) is a process known as sampling and is described in Chapter 1 (see Figures 1.5 and 1.6). Sampling is a nonlinear process and has some peculiar effects on the signal's spectrum. Figure 4.1A shows an example of a magnitude frequency spectrum of a hypothetical 1.0-second periodic, continuous signal, as determined using continuous Fourier series analysis. The fundamental frequency is $1/T = 1$ Hz and the first 10 harmonics are plotted out to 10 Hz. For this particular signal, there is little energy above 7.0 Hz.

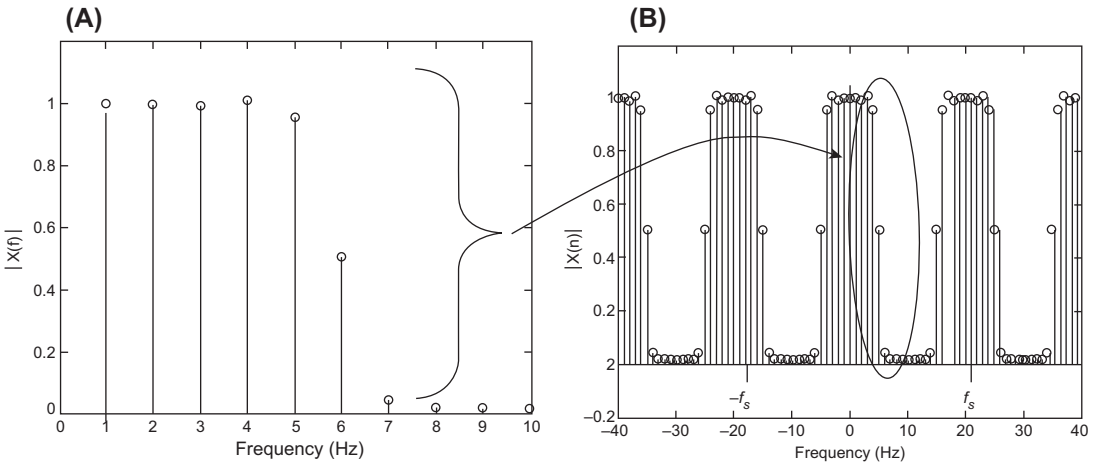

FIGURE 4.1 (A) The spectrum of a continuous signal. (B) The spectrum of this signal after being sampled at $f_s = 20$ Hz. Sampling produces a larger number of frequency components not in the original spectrum, even components having negative frequency. The sampled signal has a spectrum that is periodic at the sampling frequency (20 Hz) and has an even symmetry about 0.0 Hz, as well as symmetry about the sampling frequency, f_s. Since the sampled spectrum is periodic, it goes on forever and only a portion of it can be shown. The spectrum is just a series of points; the vertical lines are drawn to improve visualization.

If we apply the mathematics describing the sampling to the Fourier transform, we find that the sampling process produces many additional frequencies that were not in the original signal. After sampling at 20 Hz, the sampled magnitude spectrum, now plotted over a larger frequency range, is shown in Figure 4.1B. Only a portion can be shown because the sampled spectrum is, theoretically, infinite. The new spectrum is itself periodic, with a period equal to the sample frequency, f_s (in this case 20 Hz). The spectrum also contains negative frequencies (it is, after all, a theoretical spectrum). It has even symmetry about $f = 0.0$ Hz as well as about both positive and negative multiples of f_s. Finally, the portion between 0 and 20 Hz also has even symmetry about the center frequency, $f_s/2$ (in this case 10 Hz).

The spectrum of the sampled signal is certainly bizarre, but comes directly out of the mathematics of sampling as described in Chapter 5 (Section 5.7.1). When we sample a continuous signal, we effectively multiply the original signal by a periodic impulse function (with period f_s) and that multiplication process produces all those additional frequencies. Even though the negative frequencies are mathematical constructs, their effects are felt because they are responsible for the symmetrical frequencies above $f_s/2$ as clearly noted in Figure 3.21. If the sampled signal's spectrum is different from the original signal's spectrum, it stands to reason that the sampled signal is different from the original. If the sampled signal is not the same as the original and we cannot somehow link the two, then digital signal processing is a lost cause. We would be processing something unrelated to the original signal. The critical question is: given that the sampled signal is different from the original, can we find some way to reconstruct the original signal from the sampled signal? The frequency domain version of that question is: can we reconstruct the unsampled spectrum from the sampled spectrum? The definitive answer is: maybe, but it depends on things we can understand and measure.

Figure 4.2 shows just one period of the spectrum shown in Figure 4.1B, the period between 0 and f_s Hz. In fact, this is the only portion of the spectrum that can be calculated by the discrete Fourier transform (DFT); all the other frequencies shown in Figure 4.1B are theoretical (but not inconsequential). Comparing this spectrum to the spectrum of the original signal, Figure 4.1A, we see that the two are the same for the first half of the spectrum, that is, up to $f_s/2$. The second half is just the mirror image of the first half. These mirror image frequencies are just the negative frequencies reflected back from f_s.

The mirror image frequencies, those above $f_s/2$, were not part of the original signal. But the lower frequencies, those below $f_s/2$, are in the original spectrum. So if we somehow got rid of all frequencies above $f_s/2$, we would have our original spectrum. We can get rid of the frequencies above $f_s/2$ by filtering them out. Just knowing that it is possible to get back to the original spectrum is sufficient to justify our sampled computer data; we just ignore the frequencies above $f_s/2$. The frequency $f_s/2$ is so important it has its own name: the "Nyquist[1] frequency."

This strategy of just ignoring all frequencies above the Nyquist frequency ($f_s/2$) works well and is the approach that is commonly adopted. But it can be used only if the original signal does not have spectral components at or above $f_s/2$. Consider a situation in which four sinusoids with respective frequencies of 100, 200, 300, and 400 Hz are sampled at a frequency of

[1]Nyquist was one of many prominent engineers to hone his skills at the former Bell Laboratories during the first half of the 20th century. He was born in Sweden, but received his education in the United States.

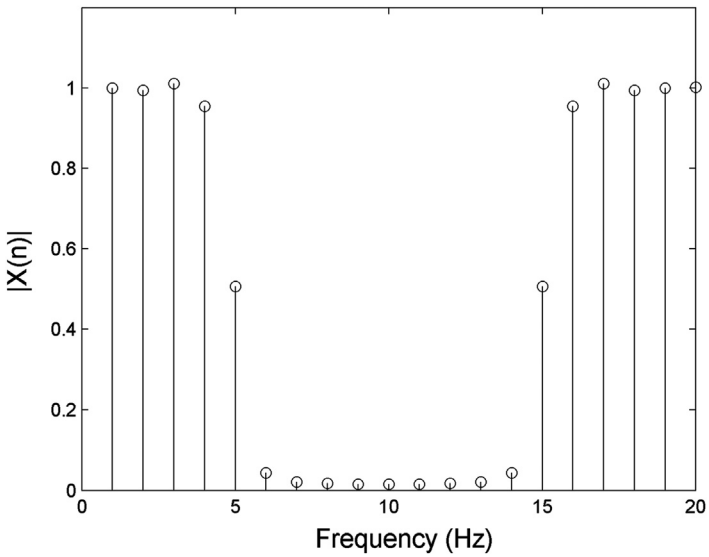

FIGURE 4.2 A portion of the spectrum of the sampled signal whose unsampled spectrum is shown in Figure 4.1A. There are more frequencies in this sampled spectrum than in the original, but they are distinct from and do not overlap the original frequencies. Separating out these unwanted spectral components through some sort of filtering should be possible.

1000 Hz. The spectrum produced after sampling actually contains eight frequencies, Figure 4.3A: the four original frequencies plus the four mirror image frequencies reflected about $f_s/2 = 500$ Hz. As long as we know, in advance, that the sampled signal does not contain any frequencies above the Nyquist frequency (500 Hz), we do not have a problem: we know that the first four frequencies are those of the signal and the second four, above the Nyquist frequency, are the reflections, which can be ignored. However, a problem occurs if the signal contains frequencies higher than the Nyquist frequency. The reflections of these high-frequency components will be reflected back into the lower half of the spectrum. This is shown in Figure 4.3B where the signal now contains two additional frequencies at 650 and 850 Hz. These frequency components have their reflections in the lower half of the spectrum: at 350 and 150 Hz, respectively. It is now no longer possible to determine if the 350 and 150 Hz signals are part of the true spectrum of the signal (i.e., the spectrum of the signal before it was sampled) or whether these are reflections of signals with frequency components greater than $f_s/2$ (which in fact they are). Both halves of the spectrum now contain mixtures of frequencies above and below the Nyquist frequency, and it is impossible to know where they really belong. This confusing condition is known as "aliasing." The only way to resolve this ambiguity is to ensure that all frequencies in the original signal are less than the Nyquist frequency.

If the original signal contains frequencies above the Nyquist frequency, then you cannot determine the original spectrum from what you have in the computer and you cannot reconstruct the original analog signal from the one in the computer. The frequencies above the Nyquist frequency have hopelessly corrupted the signal stored in the computer. Fortunately, the converse is also true. If there are no corrupting frequency components in the original signal (i.e., the signal contains no frequencies above half the sampling frequency), the spectrum in the computer can be adjusted to match the original signal's spectrum if we eliminate

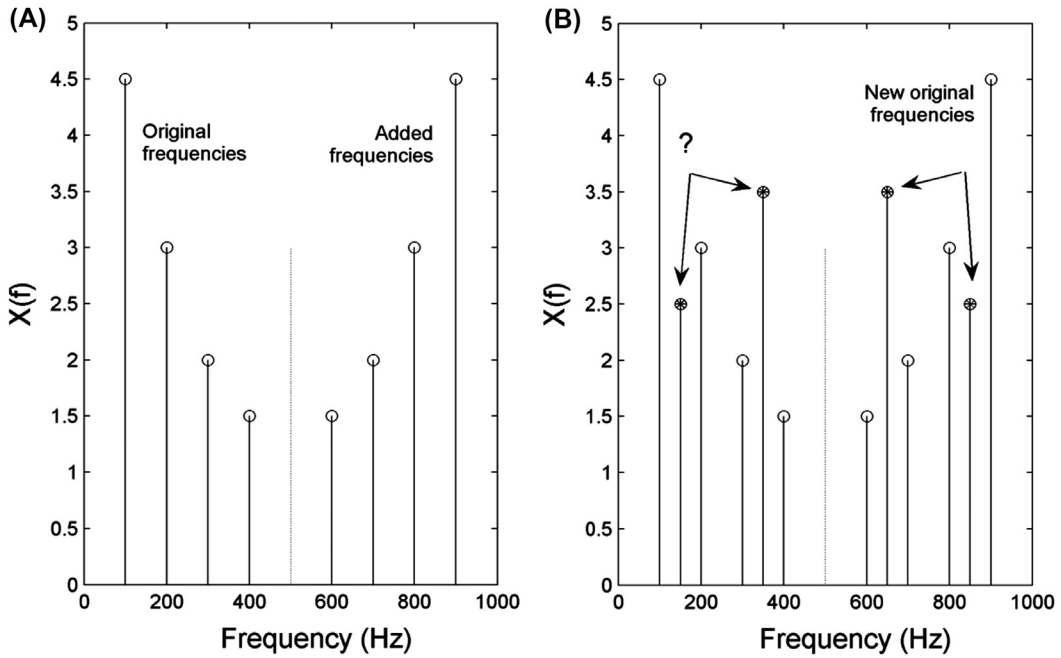

FIGURE 4.3 (A) Four sine waves between 100 and 400 Hz are sampled at 1 kHz. Sampling essentially produces new frequencies not in the original signal. The additional frequencies are a mirror image reflection around $f_s/2$, the Nyquist frequency. As long as the frequency components of the sampled signal are all below the Nyquist frequency as shown here, the upper frequencies do not interfere with the lower spectrum and can simply be ignored. (B) If the sampled signal contains frequencies above the Nyquist frequency, they are reflected into the lower half of the spectrum (filled circles). It is no longer possible to determine which frequencies belong where, an example of aliasing.

or disregard the frequencies above the Nyquist frequency. (Elimination of frequencies above the Nyquist frequency can be achieved by low-pass filtering, and the original signal can be reconstructed.) This leads to the famous "Sampling Theorem" of Shannon: the original signal can be recovered from a sampled signal provided the sampling frequency is more than twice the maximum frequency[2] contained in the original:

$$f_s > 2f_{max} \tag{4.1}$$

Usually the sampling frequency is under software control, and it is up to the biomedical engineer doing the sampling to ensure that f_s is high enough. To make elimination of the unwanted higher frequencies easier, it is common to sample at three to five times f_{max}. This increases the spacing between the frequencies in the original signal and those generated by the sampling process, Figure 4.4. The temptation to set f_s higher than is really necessary is

[2]The signal must not contain any frequencies above the Nyquist frequency. In other words, any signal frequencies above the sampling frequency, including those from noise, must have negligible energy.

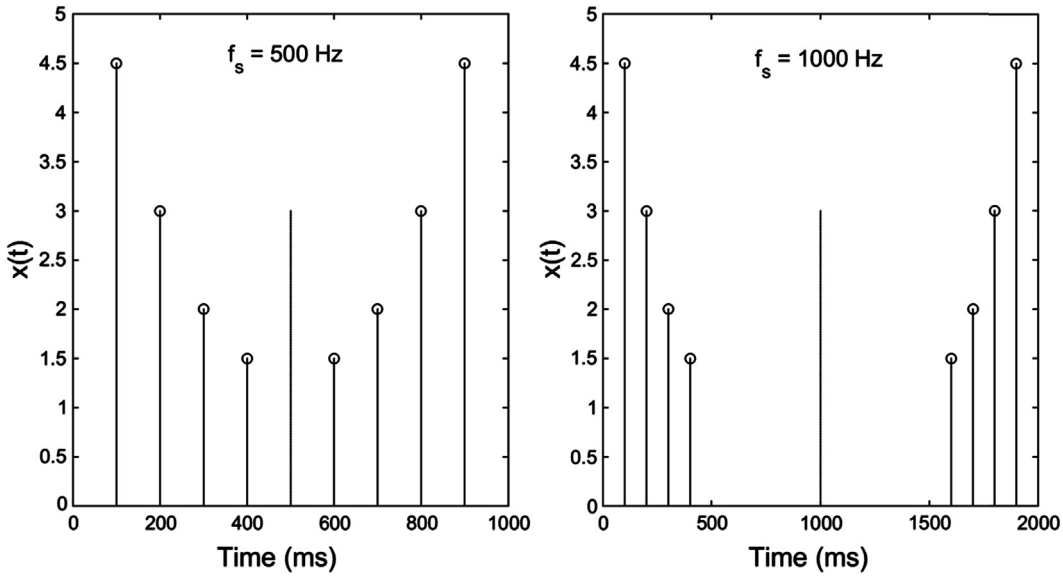

FIGURE 4.4 The same signal is sampled at two different sampling frequencies. The higher sampling frequency provides much greater separation between the original spectral components and those produced by the sampling process.

strong, and it is a strategy often pursued. However, excessive sampling frequencies lead to large data storage and processing requirements that needlessly overtax the computer system.

The concepts behind sampling and the sampling theorem can also be described in the time domain. Consider a single sinusoid. (Since all periodic waveforms can be broken into sinusoids, the influence of sampling on a single sinusoid can be extended to cover any general waveform.) In the time domain, Shannon's sampling theorem states that a sinusoid can be accurately reconstructed as long as two or more (evenly spaced) samples are taken over its period. This is equivalent to saying that f_s must greater than $2\,f_{sinusoid}$. Figure 4.5 shows a main sine wave (solid line) defined by two samples per cycle (black circles). The Shannon sampling theorem states that no other sinusoids of a lower frequency can pass through both these points, so these two samples uniquely define this sine wave. This spacing would also uniquely define any sine wave that was of a lower frequency. However, there are many higher frequency sine waves that can pass cleanly through these two points, two of which are shown in Figure 4.5 as dashed and dotted lines. The two higher frequency sine waves shown are second and third harmonics of the main sine wave. In fact, all the higher harmonics of the main sine wave would pass though these two points, so there are an infinite number of higher frequencies defined by the 2 samples. These higher frequency sine waves give rise to the added points in the sample spectrum as shown in Figure 4.1; they are the source of the additional frequencies.[3]

[3]The fact that higher frequency sinusoids are all harmonics explains why the added frequencies are themselves periodic.

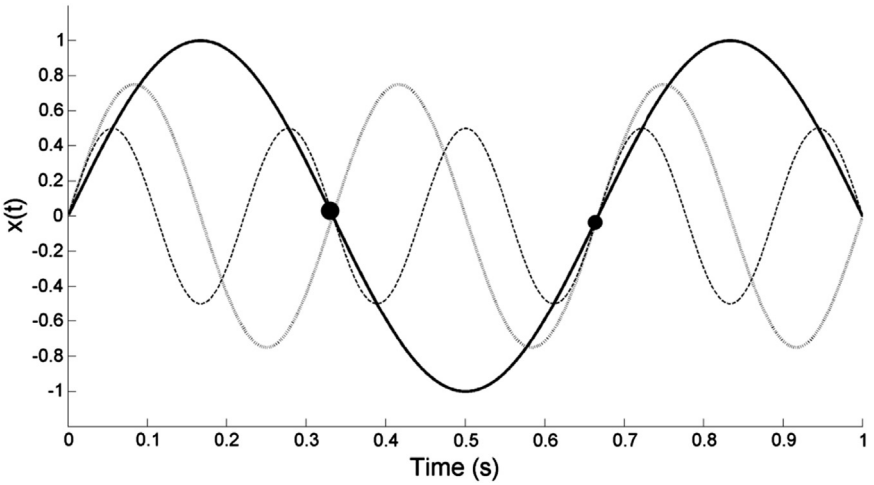

FIGURE 4.5 A sine wave (*solid line*) is sampled at two locations (*black circles*) within one period. The time domain interpretation of Shannon's sampling theorem states that no other sine wave of a lower frequency can pass though these two points. This sine wave is uniquely defined. However, an infinite number of higher frequency sine waves can pass though those two points (all the harmonics), two of which are shown. These higher frequency sine waves contribute the additional points shown on the spectrum of Figure 4.1.

Figure 4.6 illustrates aliasing using an undersampled sine wave. A 5 Hz sine wave is sampled at 7 samples/s, so $f_s/2 = 3.5$ Hz. A 2 Hz sine wave (*dotted line*) also passes through the seven points. This is predicted by aliasing: the 5 Hz sine reflected about f_s would be $7 - 5 = 2$ Hz.

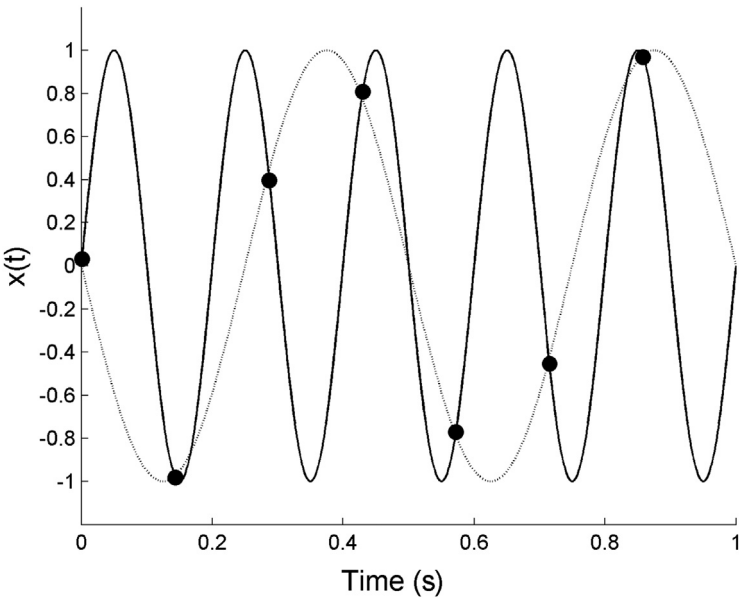

FIGURE 4.6 A 5 Hz sine wave (*solid line*) is sampled at 7 Hz (seven samples over a 1-s period). The seven samples also fit a 2 Hz sine wave, as is predicted by aliasing.

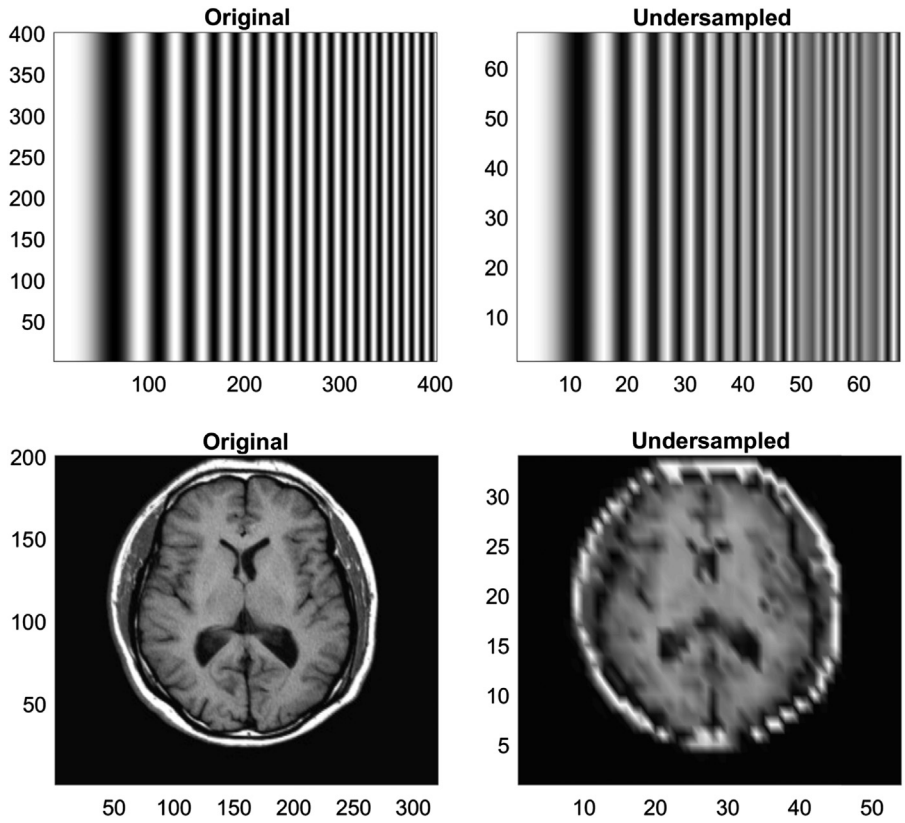

FIGURE 4.7 Two images that are correctly sampled (left side) and undersampled (right side). The upper pattern is a sine wave that increases in spatial frequencies from left to right. The undersampled image shows additional sinusoidal frequencies folded into the original pattern because of aliasing. The lower image is an MR image of the brain and the undersampled image has jagged diagonals and a moiré pattern, both characteristics of undersampled images.

Aliasing can also be observed in images. Figure 4.7 shows two image pairs: the left images are correctly sampled and the right images are undersampled. The upper images are of a sine wave that increases in spatial frequency going from left to right. This is the image version of a "chirp" signal that increases in frequency over time. The left image shows the expected smooth progression of sinusoidal bars. The right image is fine for the lower spatial frequencies, but as spatial frequency increases, additional frequencies are created because of aliasing, disrupting the progression. The lower images are MR images of the brain and the undersampled image shows jagged diagonals and a type of moiré pattern that is characteristic of undersampled images.

EXAMPLE 4.1

Construct a signal consisting of three sine waves at 100, 200, and 350 Hz. Find and plot the magnitude spectrum of this signal assuming two different sampling frequencies: $f_s = 800$ and 500 Hz. Use an N of 512 points and label and scale the frequency axes. Plot only valid (i.e., $N/2$) points.

Solution: Use a loop to operate on the two sampling frequencies. Construct a time vector and a frequency vector with the appropriate sampling frequency. Use the time vector to construct the signal as the sum of three sine waves and the frequency vector for plotting. Find the magnitude spectrum and plot only $N/2$ points.

```
%  Example 4.1 Example of aliasing.
%
N = 512;                        % Number of points, N
N2 = N/2;                       % Half N
fs = [800 500];                 % Sample frequencies
for k = 1:2
  t = (0:N-1)/fs(k);            % Time vector
  f = (1:N)*fs(k)/N;            % Frequency vector
  x = sin(2*pi*100*t) + sin(2*pi*200*t) + sin(2*pi*350*t);    % Signal
  Xmag = abs(fft(x));           % Magnitude spectrum
  subplot(1,2,k);
  plot(f(1:N2),Xmag(1:N2),'k'); % Plot magnitude spectrum
  .......labels and title........
end
```

Results: The two spectra are shown in Figure 4.8. In both cases, f_{max} is 350 Hz. When $f_s > 2\,f_{max}$ (i.e., 700 Hz), the three peaks are found at the correct frequencies: 100, 200, and 350 Hz, Figure 4.8A. When the sampling frequency is reduced to 500 Hz, f_s is now $< 2\,f_{max}$. Consequently, the peak at 350 Hz manifests as a false peak at 150 Hz, Figure 4.8B. (Note: $500 - 350 = 150$ Hz). This is a classic example of aliasing.

4.2.2 Amplitude Slicing—Quantization

By selecting an appropriate sampling frequency, it is possible to circumvent problems associated with time slicing, but what about amplitude slicing, i.e., quantization? In Chapter 1, we note that amplitude resolution is given in terms of the number of bits in the binary output with the assumption that the least significant bit in the output is accurate (which is not always true). Typical analog-to-digital converters (ADCs) feature 8-, 12-, and 16-bit outputs, with 12 bits presenting a good compromise between conversion resolution and cost. In fact, most biological signals do not have sufficient signal-to-noise ratio (SNR) to justify a higher resolution; you are simply obtaining a more accurate conversion of the noise.

The number of bits used for conversion sets an upper limit on the resolution, and determines the quantization error. The more bits used to represent the signal, the finer the resolution of the digitized signal and the smaller the quantization error. The quantization error is

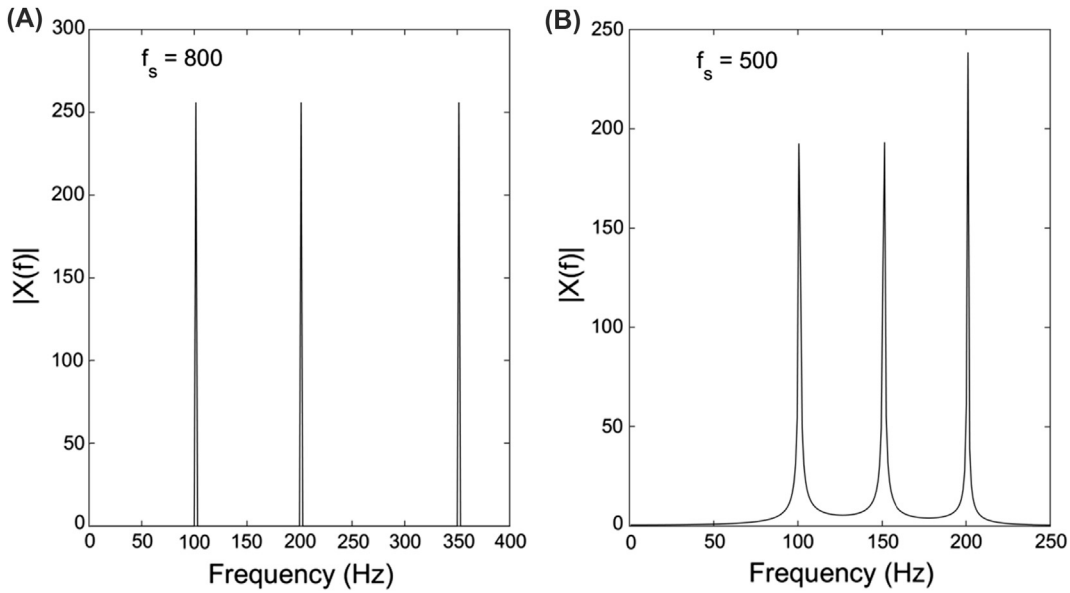

FIGURE 4.8 Magnitude spectra of a signal consisting of three sine waves at 100, 200, and 350 Hz sampled at two different frequencies: 800 and 500 Hz. (A) When $f_s = > 2\,f_{max}$, three peaks are found at the correct frequencies. (B) When $f_s = < 2\,f_{max}$, the peak that was greater than $f_s/2$ (350 Hz) appears folded back as a false peak at 150 Hz (Note: $f_s/2 - f_{max} = 500 - 350 = 150$ Hz.)

the difference between the original continuous signal value and the digital representation of that level after sampling, Figure 4.9. This error can be thought of as some sort of noise superimposed on the original signal. If a sufficient number of quantization levels exist (say $N > 64$, equivalent to seven bits), the distortion produced by quantization error may be modeled as additive, independent white noise with zero mean and a variance determined by the quantization step size, q. As described in Example 1.1, the quantization step size, q, is the maximum voltage the ADC can convert, divided by the number of quantization levels, which is $2^N - 1$; hence, $q = V_{MAX}/2^N{-}1$. The variance or mean square error can be determined using the expectation function from basic statistics:

$$s^2 \equiv \sigma^2 = \overline{\sigma^2} = \int_{-\infty}^{\infty} e^2 PDF(e)de \tag{4.2}$$

where $PDF(e)$ is the uniform probability density function and e is the error voltage (the bottom trace of Figure 4.8). The $PDF(e)$ for a uniform probability distribution that ranges between $-q/2$ to $+q/2$ is simply $1/q$. Substituting $1/q$ for $PDF(e)$ in Equation 4.2:

$$\sigma^2 = \int_{-q/2}^{q^{\delta}/2} e^2 \left(\frac{1}{q}\right) de = \left.\frac{\left(\frac{1}{q}\right)e^3}{3}\right|_{-q/2}^{q/2} = \frac{q^2}{12} = \frac{V_{MAX}^2}{12(2^N - 1)^2} \tag{4.3}$$

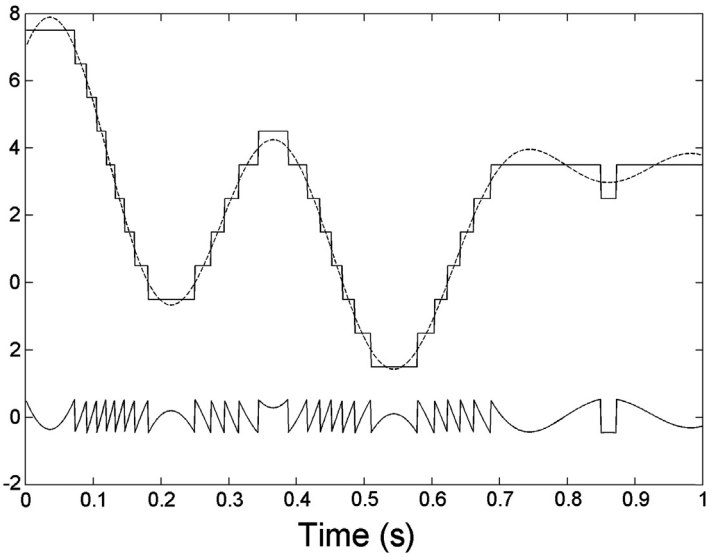

FIGURE 4.9 Quantization (amplitude slicing) of a continuous waveform. The lower trace shows the error between the quantized signal and the input.

where V_{MAX} is the maximum voltage the ADC can convert and N is the number of bits out of the ADC. Assuming a uniform distribution with zero mean for the quantization noise, the root mean square (RMS) value of the noise (Equation 2.3) would be approximately equal to the standard deviation, σ (Equation 2.12).

EXAMPLE 4.2

What is the equivalent quantization noise (RMS) of a 12-bit ADC that has an input voltage range of $\pm$ 5.0 volts? Assume the converter is perfectly accurate.

Solution: Apply Equation 4.3. Note that V_{max} would be 10 volts since the range of the ADC is from -5 to $+5$ volts.

$$\sigma^2 = \frac{V_{MAX}^2}{12\left(2^N - 1\right)^2} = \frac{10^2}{12\left(2^{12} - 1\right)^2} = \frac{100}{12(4095)^2} = 4.97 \times 10^{-7}$$

$$V_{RMS} \cong \sigma = \sqrt{4.97 \times 10^{-7}} = 0.705 \text{ mV}$$

4.2.3 Data Truncation

Given the finite memory capabilities of computers, it is usually necessary to digitize a shortened version of the signal. For example, only a small segment of a subject's ongoing electroencephalography (EEG) signal can be captured for analysis on the computer. The length of this shortened data segment is determined during the data acquisition process and is usually a compromise between the desire to acquire a large sample of the signal and the need to limit

computer memory usage or data acquisition time. Taking just a segment of a much longer signal involves truncation of the signal, and this has an influence on the spectral resolution of the stored signal. As described next, both the number of data points and the manner in which the signal is truncated will alter the spectral resolution.

4.2.3.1 Data Length and Spectral Resolution

As discussed in Chapter 3, all digitized signals are assumed to be periodic. The period, T, is the time length of the data segment. Recall that the Fourier series frequency components depend on this period, specifically:

$$f_m = \frac{m}{T} = mf_1 \tag{4.4}$$

where m is the harmonic number, T is the period, and f_1 equals $1/T$.

For a discrete signal of N samples, the equivalent time duration, T, is N times the sample interval, T_s or N divided by the sample frequency, f_s:

$$T = \frac{N}{f_s} \tag{4.5}$$

and the equivalent frequency of a given harmonic number, m, is:

$$f_m = \frac{m}{T} = \frac{m}{\frac{N}{f_s}} = \frac{mf_s}{N} \tag{4.6}$$

The last data point from the DFT has a frequency value of f_s since:

$$f_{\text{Last}} = f|_{m=N} = \frac{Nf_s}{N} = f_s \tag{4.7}$$

The frequency resolution of a spectral plot is the difference in frequency between harmonics, which according to Equation 4.6[4] is f_s/N:

$$f_{\text{Resolution}} = \frac{f_s}{N} \tag{4.8}$$

Just as with the continuous Fourier transform, frequency resolution of the DFT depends on the period (i.e., time length) of the data. For the DFT, the resolution is equal to f_s/N, from Equation 4.8. So, for a given sampling frequency, the more samples (N) in the signal, the smaller the frequency increment between successive DFT data points. The more points sampled, the higher the spectral resolution. Of course you could also reduce f_s, but this strategy is limited by the sampling theorem, Equation 4.1.

[4]This is essentially the same equation we use in MATLAB programs to generate the horizontal axis used in frequency plots (i.e., in MATLAB, f = (0:N−1)*fs/N).

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. We used this approach, zero padding, to extend signals when computing cross-correlation in Chapter 2. Using it here to extend the length of a signal may sound like cheating, but we could argue that we cannot really know what the signal was doing outside of the data segment we have, and maybe it really was zero. Other, more complicated padding techniques can be used, such as extending the last values of the signal on either end, but zero padding is by far the most common strategy for extending data.

Zero padding gives the appearance of a spectrum with higher resolution. Example 4.3 shows that zero padding certainly appears to improve the resolution of the resulting spectrum, producing a spectrum with more points that better illustrate the spectrum's details. Of course, artificially extending the period with zeros does not increase the information in the spectrum and the spectral resolution is really not any better. But zero padding provides an interpolation of the points that were 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. Nonetheless, the interpolated spectrum certainly looks better when plotted.

EXAMPLE 4.3

Generate a waveform with a period $T = 1.0$ s that is a triangle wave for the first 0.5 s and zero for the rest of the period. Assume $f_s = 100$ Hz so $N = 100$ points. Calculate and plot the magnitude spectrum. Zero pad the signal to extend the period to 2 and 6 s and recalculate and plot the power spectrum. Limit the spectral plot to a range of 0−20 Hz.

Solution: Generate the 1.0 triangle waveform. Since $f_s = 100$ Hz, the original signal should be padded by 100 and 500 additional points to extend the period to 2 and 6 s. Calculate the spectrum using `fft`. Use a loop to calculate and plot the magnitude spectra of the three signals.

```
% Example 4.3  Example of zero padding on apparent spectral resolution
%
fs = 100;                                % Sampling frequencies
N1 = [0 100  500];                       % Zero padding numbers
x =[ (0:25) (24:-1:0),zeros(1,49) ];     % Generate triangle signal, 100 pts
for k = 1:3
  y = [x zeros(1,N1(k))];                % Zero pad signal with 0, 100, and 500
  N = length(y);                         % Get data length
  t = (0:N-1)/fs;                          % Construct time vector for plotting
  f = (0:N-1)*fs/N;                      % Construct frequency vector for plotting
  subplot(3,2,k*2-1);
  plot(t,y,'k');                         % Plot the signal
  ..... Labels and titles.....
  subplot(3,2,k*2);
  Y = abs(fft(y));                       % Calculate the magnitude spectrum
  plot(f, Y,'.k');                       % Plot spectrum using individual points
  .....Labels and titles.....
end
```

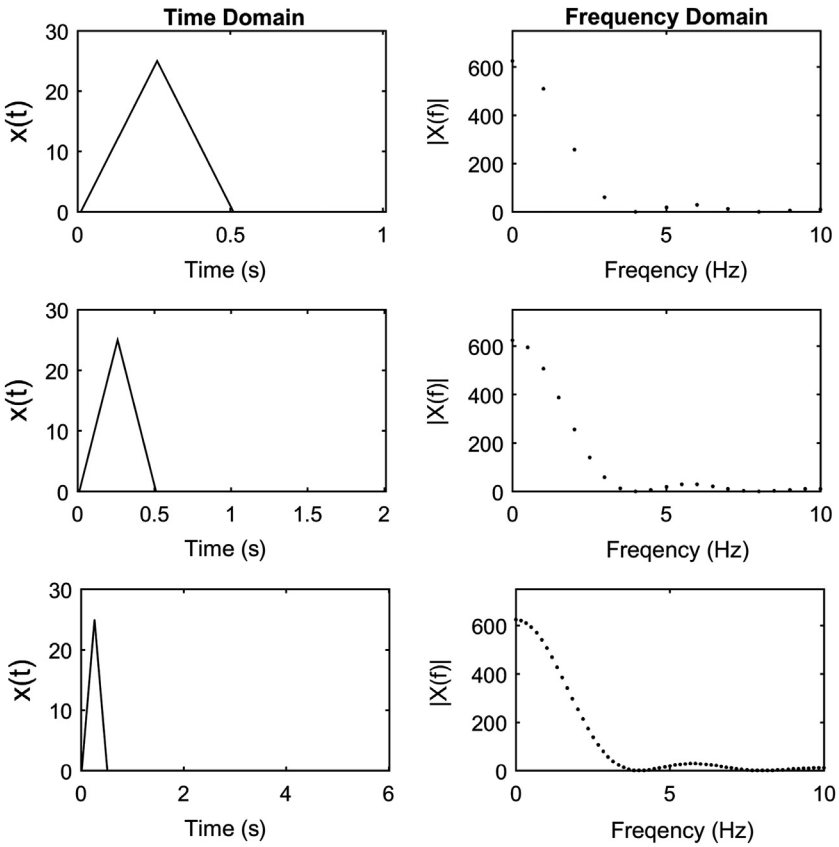

FIGURE 4.10 A waveform having an actual period of 1.0 s (upper left) and its associated frequency spectrum (upper right). Extending the period to 2 and 6 s by adding zeros decreases the spacing between the frequency points, producing smoother looking frequency curves (middle and lower plots).

Results: The time and magnitude spectrum plots are shown in Figure 4.10. The spectral plots all have the same shape, but the points are more closely spaced with the zero-padded data. As shown in Figure 4.10, simply adding zeros to the original signal produces a better looking curve, which explains the popularity of zero padding (even if no additional information is produced). MATLAB makes zero padding during spectral analysis easy, as the second argument of the fft routine (i.e., `fft(x,N)`) adds zeros to the original signal if the original signal length is less than `N`. (If `N` is less than the data length, the data are truncated to that length.)

Taking the DFT implicitly assumes that the signal is periodic with a period of $T = N/f_s$ (Equation 4.5). The frequency spectrum produced is a set of numbers spaced f_s/N apart (Equation 4.8). However, if we make the assumption that our signal is actually aperiodic (i.e., that $T \rightarrow \infty$), then the spacing between points goes to zero and the points become a continuous line. This is the assumption that is commonly made when we plot the spectrum

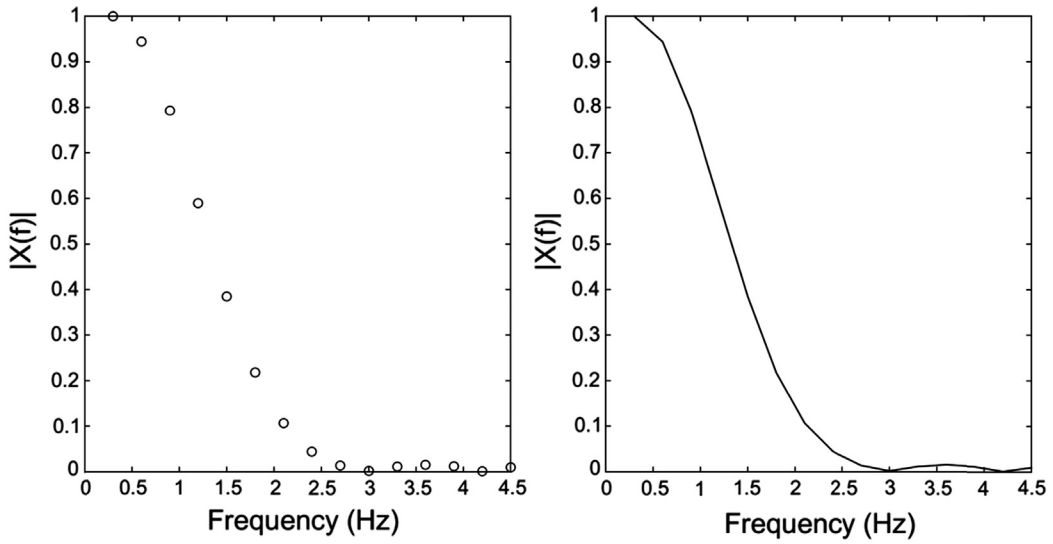

FIGURE 4.11 (A) The discrete Fourier transform assumes the signal is periodic and computes a spectrum consisting of discrete points. (B) Making the assumption that the signal really is aperiodic with an infinite period justifies connecting the discrete points forming a smooth curve. This assumption is commonly made when plotting frequency spectra.

not as a series of points, but as a smooth curve, Figure 4.11 (right side). Although this is commonly done when plotting spectra, you should be aware of the underlying truth: an implicit assumption is being made that the signal is aperiodic and, although the calculated spectrum is really a series of discrete points, they are joined together because of this assumption.

4.2.4 Data Truncation—Window Functions

The third limitation of signal digitization is the usual need to truncate the original real-world signal because of memory constrains. Truncation can be thought of as a "windowing" process whereby the original data are multiplied by a finite-length function that limits the data length. For example, often the original signal is simply cut at two time points to extract the digitized signal. We could think of this extraction as multiplying the original signal by a rectangular waveform that is 1.0 over the length of the digitized signal and 0.0 everywhere else, Figure 4.12. Note that this approach usually produces abrupt changes or discontinuities at the endpoints.

The discontinuities at the endpoints can produce artifacts in the spectrum, particularly if the data length is small. One way to reduce these discontinuities would be to impose a window function of our own invention that tapers the data at the ends to zero. No need to invert such a window as many such tapering window shapes have been developed. An example of one such tapering window is the "Hamming window" shown in Figure 4.13. This window is often used in MATLAB routines that are likely to involve short data sets. The Hamming

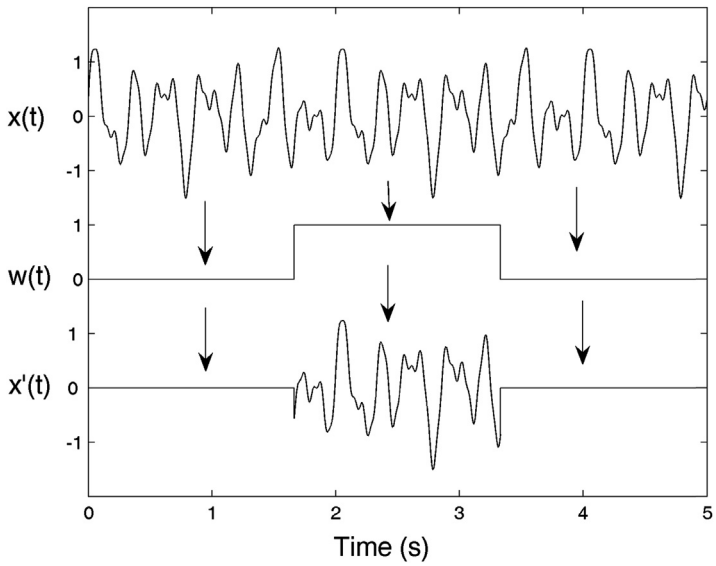

FIGURE 4.12 Data truncation or shortening can be thought of mathematically as multiplying the original data, $x(t)$ (*upper curve*), by a rectangular window function, $w(t)$ (*middle curve*), to produce the truncated data that are in the computer, $x'(t)$ (*lower curve*). The window function has a value of 1.0 over the length of the truncated data and 0.0 everywhere else. Note that this window function generally results in abrupt changes at the endpoints.

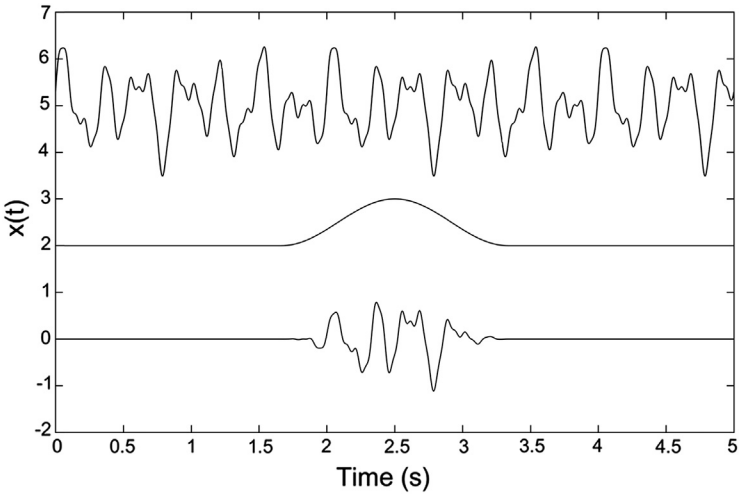

FIGURE 4.13 The application of a Hamming window to shape the data truncated by a rectangular window in Figure 4.12. This window is similar to a half sine function and, when multiplied with the data set, tapers the two ends of the signal toward zero. The application of such a tapering window decreases the resultant spectral resolution, but also reduces the influence of noise on the spectrum.

window has the shape of a raised half sine wave, which when applied to the truncated signal reduces endpoint discontinuities:

$$w[n] = 0.5 - 0.46\cos\left(\frac{2\pi n}{N}\right) \tag{4.9}$$

where $w[n]$ is the Hamming window function and N is the window (and data) length. The variable n ranges between 1 and $N + 1$.

Multiplying your signal by a function like that shown in Figure 4.13 may seem a bit extreme and it is controversial in the signal processing community. Moreover, the benefit of windows such as the Hamming window is generally quite subtle. Except for very short data segments, a rectangular window (i.e., simple truncation and no additional windowing) is usually the best option. Applying the Hamming window to the data will result in a spectrum with slightly better noise immunity, but overall frequency resolution is reduced. Figure 4.14 shows the magnitude spectra of two closely spaced sinusoids with noise that were obtained using the FFT after application of two different windows: a rectangular window (i.e., simple truncation) and a Hamming window. The spectra are really quite similar, although the spectrum produced after applying a Hamming window to the data shows a slight loss in resolution as the two frequency peaks are not as sharp. The peaks due to background noise are also somewhat reduced by the Hamming window.

If the data set is fairly long (perhaps 256 points or more), the benefits of a nonrectangular window are slight. Figure 4.15 shows the spectra obtained from a signal containing two closely spaced sinusoids (100 and 120 Hz) in noise (SNR $= -12$ dB) with and without the Hamming window. The resulting spectra are seen to be nearly the same except for a scale difference produced by the Hamming window. MATLAB's Signal Processing Toolbox contains routines to generate 17 different window functions of varying complexity. Semmlow

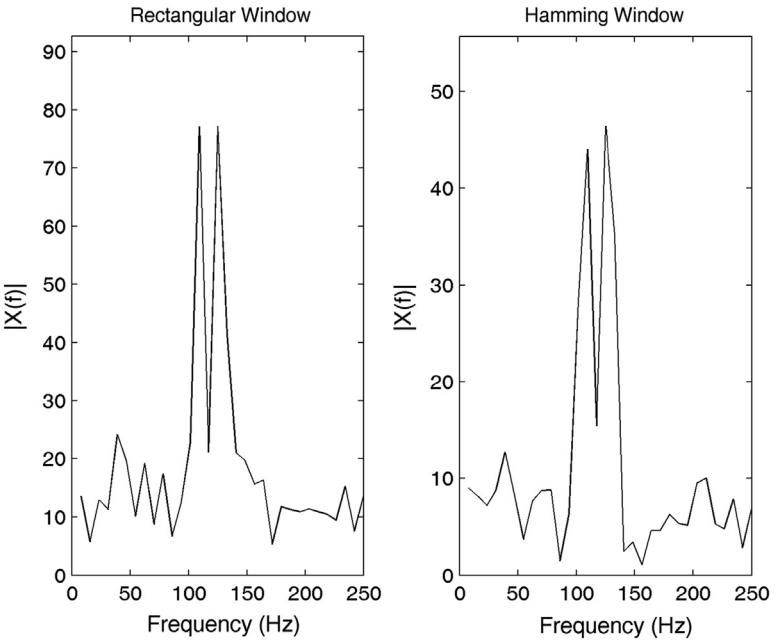

FIGURE 4.14 The spectrum from a short signal ($N = 128$) containing two closely spaced sinusoids (100 and 120 Hz) with added noise. A rectangular and a Hamming window have been applied to the signal to produce the two spectra. The spectrum produced after application of the Hamming window shows a slight loss of spectral resolution, as the two peaks are not as sharp; however, background spectral peaks due to noise have been somewhat reduced.

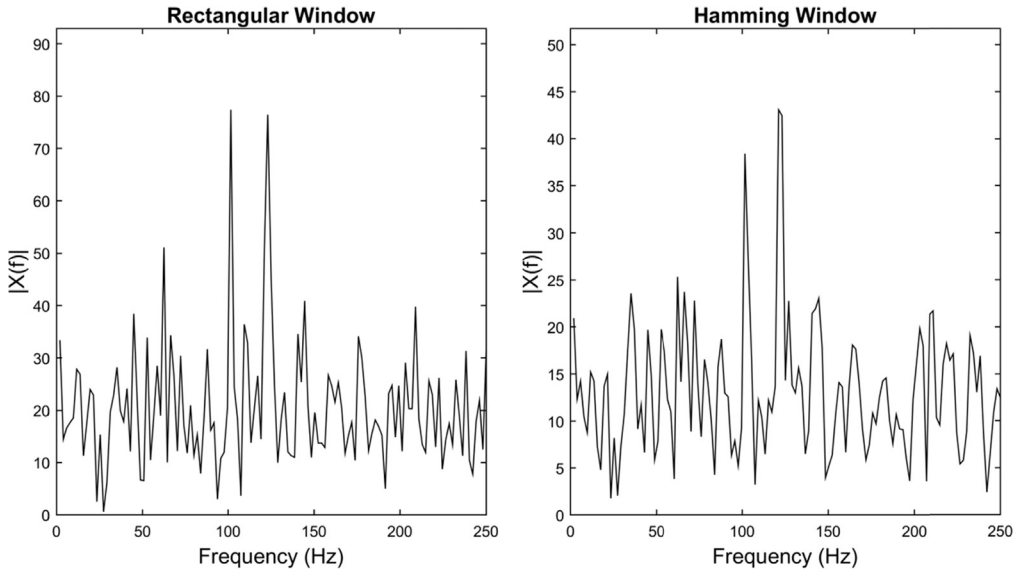

FIGURE 4.15 A spectrum from a relatively long signal ($N = 512$) containing two closely spaced sinusoids (100 and 120 Hz) in noise (SNR = −12 dB). Rectangular and Hamming windows have been applied to the signal. When the data set is long, the window function has little effect on the resultant spectrum and you might as well use a rectangular window, i.e., simple truncation.

(2014) provides a thorough discussion of the influence of various nonrectangular window functions, but the bottom line is that they typically have little effect.

4.3 POWER SPECTRUM

The power spectrum is commonly defined as the Fourier transform of the autocorrelation function. In continuous and discrete notations the power spectrum equation becomes:

$$PS(f) = \frac{1}{T}\int_0^T r_{xx}(t)e^{-j2\pi m f_1 t}dt \qquad m = 0,1,2,3\ldots \tag{4.10}$$

$$PS[m] = \sum_{n=1}^{N} r_{xx}[n]e^{-\frac{j2\pi mn}{N}} \quad m = 0,1,2,3\ldots N \tag{4.11}$$

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 two equations can be simplified to include only real cosine terms:

$$PS[m] = \sum_{n=0}^{N-1} r_{xx}[n]\cos\left(\frac{2\pi nm}{N}\right) \quad m = 0,1,2,3\ldots N \tag{4.12}$$

$$PS(f) = \frac{1}{T}\int_0^T r_{xx}(t)\cos(2\pi mft)dt \quad m = 0, 1, 2, 3, \ldots \tag{4.13}$$

Equations 4.12 and 4.13 are sometimes referred to as "cosine transforms."

Nowadays, these equations are principally of theoretical, or perhaps historical, value. We now use the direct approach to calculate the power spectrum. This 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:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt \tag{4.14}$$

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

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X(f)|^2 df \tag{4.15}$$

Equations 4.14 and 4.15 show that the magnitude spectrum squared, $|X(f)|^2$, has the same energy as the time signal. Accordingly the magnitude spectrum squared is referred to as the "energy spectral density," or more commonly, the "power spectral density" or simply the "power spectrum" (PS). So in the direct approach, the power spectrum is just the magnitude squared of the Fourier transform (or Fourier series):

$$PS(f) = |X(f)|^2 \tag{4.16}$$

This direct approach of Equation 4.16 has displaced the cosine transform for determining the power spectrum because of the efficiency of the fast Fourier transform.[5] One of the problems at the end of this chapter compares the power spectrum obtained using the direct approach of Equation 4.16 with the traditional cosine transform method represented by Equation 4.11 and, if done correctly, shows them to be identical.

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

An example of the descriptive properties of the power spectrum is given using the heart rate data shown in Example 2.19. The heart rates showed major differences in the mean and standard deviation between meditative and normal states. Applying the autocovariance to the meditative heart rate data suggested a possible repetitive structure for the variation in

[5]A variation of the cosine transform approach is still used in some advanced signal processing techniques involving simultaneous time and frequency transformation.

heart rate (Problem 29 in Chapter 2). The next example uses the power spectrum to search for structure in the frequency characteristics of both normal and meditative heart rate data.

EXAMPLE 4.4

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

Solution: The power spectrum can be determined using the direct method given in Equation 4.16. However, the heart rate data should first be converted to evenly sampled time data, and this is a bit tricky. Nonetheless, it is important to learn how to deal with such data, as unevenly sampled data are often found in studies of cardiac and neural function.

The data set obtained by a download from the PhysioNet data base provides the heart rate in beats per minute at unevenly spaced times, whenever a heartbeat occurred. The (uneven) sample times are provided as a second vector. These rate data need to be rearranged into evenly spaced time samples. This process, known as "resampling," constructs a vector of the heart rate at some selected sample interval. This evenly timed sampled vector is constructed 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. Details can be found in the MATLAB help file for `interp1`.

In the following program, the uneven x-y pairs for normal conditions are in time vector `t_pre` and heart rate vector `hr_pre`, both found in the MATLAB file `Hr_pre`. For meditative conditions, the vectors are `t_med` and `hr_med` in file `Hr_med`. To convert the heart rate data to a sequence of evenly spaced points in time, a time vector, `xi`, is first created. This vector increases in increments of 0.01 s ($t_s = 1/f_s = 1/100$) between the lowest and highest values of time (rounded appropriately) in the original data. A 100 Hz resampling frequency was chosen because this is common in HRV studies that use certain nonlinear methods, but in this example, a wide range of resampling frequencies give the same result. Evenly spaced data are produced in vector `yi` using the MATLAB interpolation routine `interp1`. Since we want the power spectrum of HRV, not heart rate per se, we subtract out the average heart rate before evaluating the power spectrum. (In Example 2.19, we used autocovariance, which automatically subtracts the mean.)

After interpolation and removal of the mean heart rate, the power spectrum is determined using `fft` and taking the square of the magnitude component. The frequency plots are limited to a range between 0.0 and 0.15 Hz since this is where most of the spectral energy is to be found.

```
% Example 4.4
% 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
```

```
%
Tmin = ceil(t_pre(1));                  % Initial time (rounded upward)
Tmax = floor(t_pre(end));               % Final time (rounded downward);
t = (Tmax:Ts:Tmin);                     % Evenly-spaced time vector
yi = interp1(t_pre,hr_pre,xi');         % Interpolate
yi = yi - mean(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. 4.16)
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
```

Results: The power spectrum of normal HRV is low and decreases with frequency, showing little energy above 0.1 Hz, Figure 4.16A. The meditative state, Figure 4.16B, shows large peaks at around 0.1−0.12 Hz, indicating that some resonant process is active at these frequencies, frequencies corresponding to a time frame of around 10 s. Feel free to speculate on the cause of this change in the heart rate rhythm.

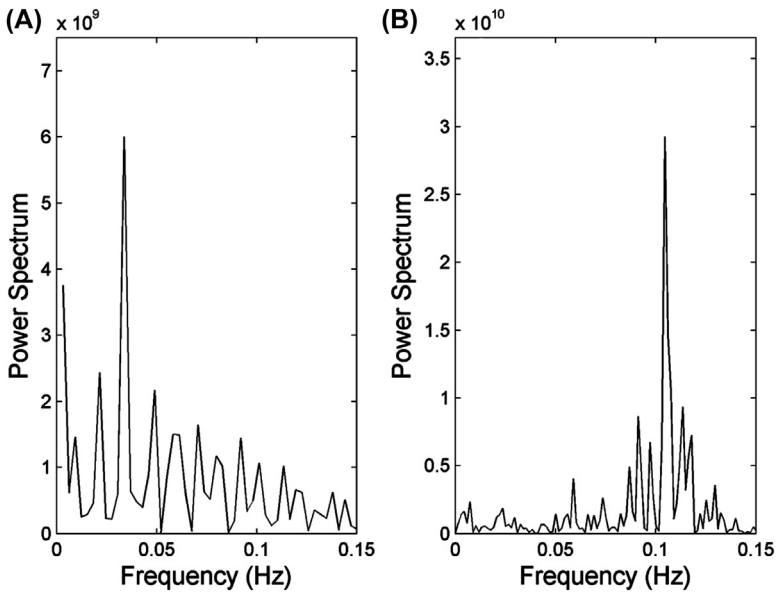

FIGURE 4.16 Power spectra of heart rate variability in the frequency range between 0 to 0.15 Hz where most of the energy is located. (A) Power spectrum of heart rate variability under normal conditions. The power decreases with frequency. (B) Power spectrum of heart rate variability during meditation. Strong peaks in power around 0.12 Hz are seen, indicating that much of the variation in heart rate is organized around these frequencies. Note the larger scale of the meditative power spectrum.

Analysis of HRV is an area of considerable current research. Spectral analysis is a commonly used tool in HRV studies. Many physiological processes influence the heart rate and the feedback nature of these processes can produce unique oscillations that can be seen in the HRV spectrum. HRV has been shown to be of diagnostic value in myocardial infarction and other heat diseases, and also in diabetes, neural disorders, sepsis, sudden infant death syndrome, depression, and other psychological disorders.

4.4 SPECTRAL AVERAGING

Although the power spectrum 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 better represents the broadband, or global features of the spectrum. 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 power spectrum is based on a direct application of the Fourier transform 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 power spectrum of each of these segments, Figure 4.17. The final spectrum is constructed from the ensemble average of the power

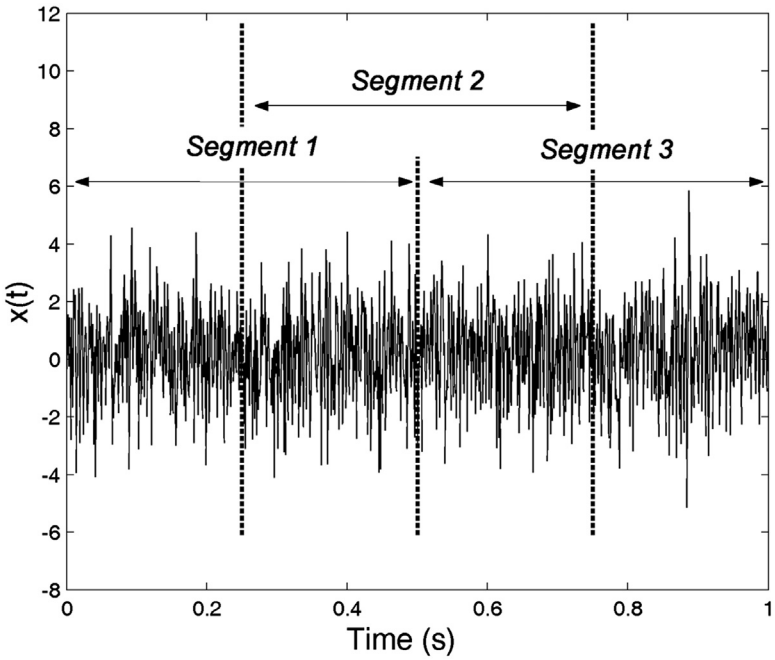

FIGURE 4.17 A noisy waveform is divided into three segments with a 50% overlap between each segment. In the Welch method of spectral analysis, the power spectrum of each segment is computed separately and an average of the three transforms gives the final spectrum.

spectra obtained from each segment.[6] Note that this averaging approach can only be applied to the power spectrum or magnitude spectrum because these spectra are not sensitive to time translation as shown in Example 3.11. Applying this averaging technique to the standard Fourier transform would not make sense because the phase spectrum would be sensitive to segment position. Averaging phases obtained for 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 this approach, overlapping segments are used and a shaping window (i.e., a nonrectangular window) is sometimes applied to each segment. Periodograms obtained from noisy data traditionally average spectra from half-overlapping segments: segments that overlap by 50%. Higher amounts of overlap have been recommended in applications when computing time is not a factor. Maximum overlap occurs when the segment is shifted by only one sample.

If we segment a signal, the length of each segment is obviously less than that of the original unsegmented signal. Each segment will have fewer samples, N. Equation 4.8 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 increased 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.

EXAMPLE 4.5

Evaluate the influence of averaging on power spectrum estimation as applied to a signal consisting of broadband and narrowband signals with added noise. The broadband signal was constructed by taking white noise and filtering it to remove frequencies above 300 Hz; then two closely spaced sinusoids were added at 390 and 410 Hz. Determine both the averaged and unaveraged spectra. The signal is in vector `x` in file `broadband1.mat` and $f_s = 1$ kHz.

Solution: Load the file `broadband1` containing the signal with narrowband and broadband components. First, calculate and display the unaveraged power spectrum. Then apply power spectrum averaging using an averaging routine. Use a segment length of 128 points and the maximum overlap of 127 points. To implement averaging, write a routine called `welch` that takes in the data, segment size, and the number of overlapping points and produces the averaged power spectrum. The main routine is:

```
% Example 4.5 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
N = length(x);              % Find signal length
```

[6]Section 2.2.4 describes ensemble averaging to reduce noise in the time domain.

```
N2 = round(N/2);            % Half data length
nfft = 128;                       % Segment size for averaging
f = (1:N)*fs/N;                   % Frequency vector for plotting
PS = abs((fft(x)).^2)/N;          % Calculate un-averaged PS
subplot(1,2,1)
plot(f(1:N2),PS(1:N2));           % Plot un-averaged Power Spectrum
  ..........labels and title.........
%
[PS_avg,f_avg]=welch(x,nfft,nfft-1,fs);       %Calculate periodogram,max.overlap
%
subplot(1,2,2)
  plot(f_avg,PS_avg);             % Plot periodogram
  .......labels and title.......
```

The `welch` routine takes in the sampling frequency, an optional parameter, which is used to generate a frequency vector useful for plotting. MATLAB power spectrum routines generally include this feature and we need to make our programs as good as MATLAB's. This program outputs only the nonredundant points of the power spectrum (i.e., up to $f_s/2$) along with the frequency vector. Finally, this routine checks the number of arguments and uses defaults if necessary, another common MATLAB feature. The program begins with a list and description of input and output parameters, then sets up defaults as needed. This is the text that is printed when someone types in "`help welch`."

```
function [PS,f] = welch(x,nfft,noverlap,fs);
  ......descriptive comments...........
%
N = length(x);           % Get data length
nfft = round(nfft);      % Make sure nfft is an interger
nfft2 = round(nfft/2);   % Half segment length
if nargin < 4            % Check arguments
  fs = 2*pi;             % Default fs
end
if nargin < 3 || isempty(noverlap) == 1
  noverlap = nfft2;      % Set default overlap at 50%
end
noverlap = round(noverlap);         % Make sure noverlap is an interger
%
%  Defaults complete. The routine now calculates the appropriate number of points
%  to shift the window and the number of averages that can be done given
%  the data length (N), window size (nff) and overlap (noverlap).
%
f = (1:nfft2)* fs/(nfft);           % Calculate frequency vector
increment = nfft - noverlap;        % Calculate window shift
nu_avgs = round(N/increment);       % Determine the number of segments
%
% Now shift the segment window and calculate the PS using Eq. 4.17
for k = 1:nu_avgs                   % Calculate spectra for each data point
```

```
    first_point = 1 + (k - 1) * increment;
    if (first_point + nfft -1) > N          % Check for possible overflow
      first_point = N - nfft + 1;           % Shift last segment to prevent overflow
    end
    data = x(first_point:first_point+nfft-1); % Get data segment
      %  MATLAB routines would add a non-rectangular window here, the default being
      %  a Hamming window.  This is left as an exercise in one of the problems
    if k == 1
      PS = abs((fft(data)).^2);             % Calculate PS, first segment
    else
      PS = PS + abs((fft(data)).^2);        % Calculate PS, add to average
    end
  end
  % Scale average and remove redundant points.  Also do not include DC term
  PS = PS(2:half_segment+1)/(nu_avgs*nfft2);
```

Analysis: This routine first checks if the sampling frequency and desired overlap are specified. If not, the routine sets f_s to 2π and the overlap to a default value of 50% (i.e., half the segment length: `nfft2`). These are the values that MATLAB uses in a similar routine called `pwelch` found in the Signal Processing Toolbox. Argument names `nfft` and `noverlap` are also lifted from the `pwelch` routine. Next, a frequency vector, `f`, is generated from `1` to `fs/2` (or π). The number of segments to be averaged is determined based on the segment size (`nfft`) and the overlap (`noverlap`). A loop is used to calculate the power spectrum using the direct method of Equation 4.16, and individual spectra are summed. Finally, the power spectrum is shortened to eliminate redundant points and the DC term, and then normalized by both the segment length and the number of spectra to generate an average spectrum.

This example determines both the unaveraged and averaged power spectra. For the averaged spectrum or periodogram, a segment length of 128 (a power of two) is specified as is the maximal overlap (an overlap equal to the segment length $N - 1$). In practice, the selection of segment length and averaging strategy is usually based on experimentation with the data.

Results: In the unaveraged power spectrum Figure 4.18A, the two sinusoids at 390 and 410 Hz are clearly seen; however, the broadband signal is noisy and poorly defined. The periodogram in Figure 4.18B is much smoother, better reflecting the constant energy in white noise, but the loss in frequency resolution is apparent as the two sinusoids are hardly visible. This demonstrates one of those all-so-common engineering compromises. Spectral techniques that produce a good representation of global characteristics such as broadband features are not good at resolving narrowband or local features such as sinusoids and vice versa.

EXAMPLE 4.6

Determine and plot the frequency characteristics of HRV during both normal and meditative states using spectral averaging. Divide the time data into eight segments and use a 50% overlap (the default). Plot using both a linear vertical axis and an axis scaled in decibel (dB).

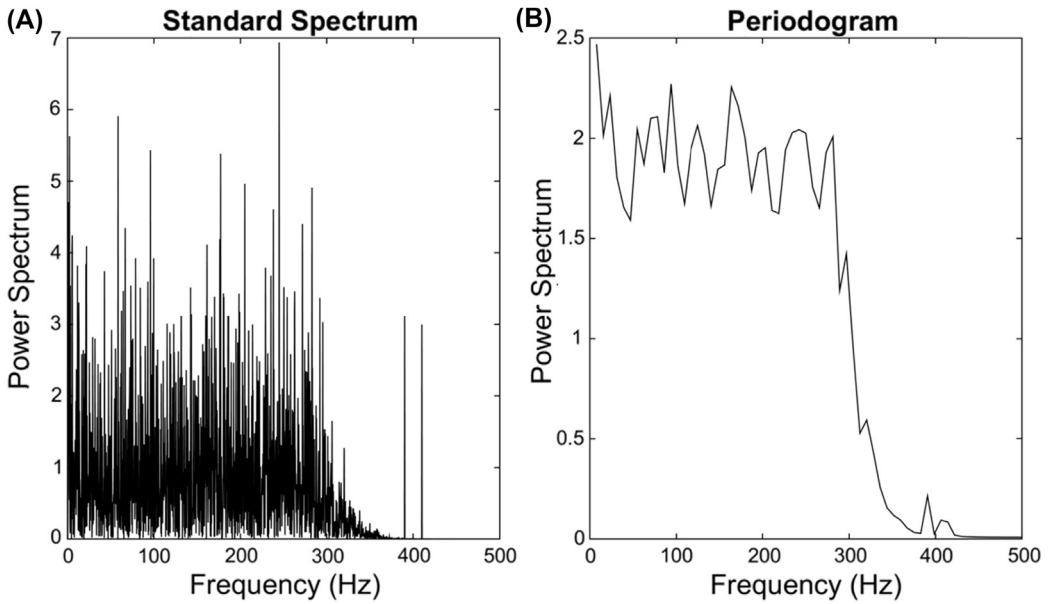

FIGURE 4.18 Power spectra obtained from a waveform consisting of a broadband signal having constant energy between 0 and 300 Hz and two sine waves at 390 and 410 Hz. (A) The unaveraged spectrum clearly shows the 390 and 410 Hz components, but the features of the broadband signal are unclear. (B) The averaged spectrum barely shows the two sinusoidal components, but produces a smoother estimate of the broadband spectrum, which is expected to be flat up to around 300 Hz. In the next example, averaging is used to estimate the power spectrum of the normal and meditative heart rate variability data.

Solution: Construct the time data by resampling at 100 Hz as was done in Example 4.4. Find the segment length by dividing the total length by eight and round this length downward. Use the `welch` routine with the default 50% overlap to calculate the average power spectrum and plot.

```
% Example 4.6 Influence of spectral averaging on heart rate data.
...... Data loading and reorganization as in Example 4.4 .......
%
nfft = floor(length(yi)/8);                    % Calculate segment length
[PS_avg,f] = welch(yi,nfft,[ ],fs);            % Periodogram, 50% overlap
PS_avg_dB = 20*log10(PS_avg);                  % Put in dB
  .......plot, label and axis.......
  ....... Repeat for meditative data ........
```

Results: The `welch` routine developed in the last example is used to calculate the average power spectrum. The sample segment length with a 50% overlap is chosen empirically, as it produces smooth spectra without losing the peak characteristic of the meditative state.

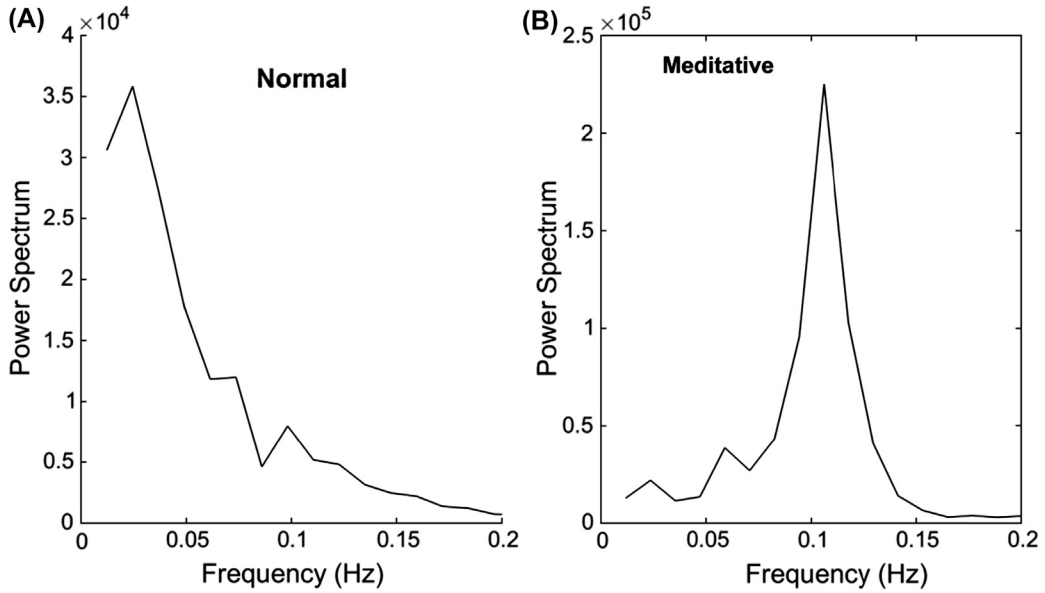

FIGURE 4.19 Power spectra taken from the same heart rate variability data used to determine the spectra in Figure 4.16, but constructed using an averaging process. The spectra produced by averaging are considerably smoother. (A) Normal conditions; (B) meditative conditions.

The results in Figure 4.19 show much smoother spectra than those of Figure 4.16, but they also lose some of the detail. The power spectrum of HRV taken under normal conditions now appears to decrease smoothly with frequency. It is common to plot power spectra in dB. Since we are dealing with power, we apply a variation of Equation 2.5: $PS_{\mathrm{dB}} = 10 \log (PS)$. (We use 10 log instead of 20 log because the square of the magnitude spectrum was taken in the calculation of the power spectrum; see Equation 4.16.) Replotting in dB shows that the normal spectrum actually decreases linearly with frequency (Figure 4.20A). The principle feature of the dB plot of meditative data is the large peak at 0.12 Hz (Figure 4.20B). Note that this peak is also present, although much reduced, in the normal data (Figure 4.20A). This suggests that meditation enhances a process that is present in the normal state as well, some sort of feedback process with an 8-s delay (1/0.12). Note that the linear decrease in power with frequency and the small peak at 0.12 Hz are not apparent in the previous spectral plots of these EEG data. This demonstrates the importance of presenting your data appropriately.

4.5 SIGNAL BANDWIDTH

The concept of representing a signal in the frequency domain brings with it additional concepts related to a signal's spectral characteristics. One of the most important of these new concepts is signal bandwidth. In Chapter 6, we extend the concept of bandwidth to systems as well as signals and find that the concepts are the same. The modern everyday use of

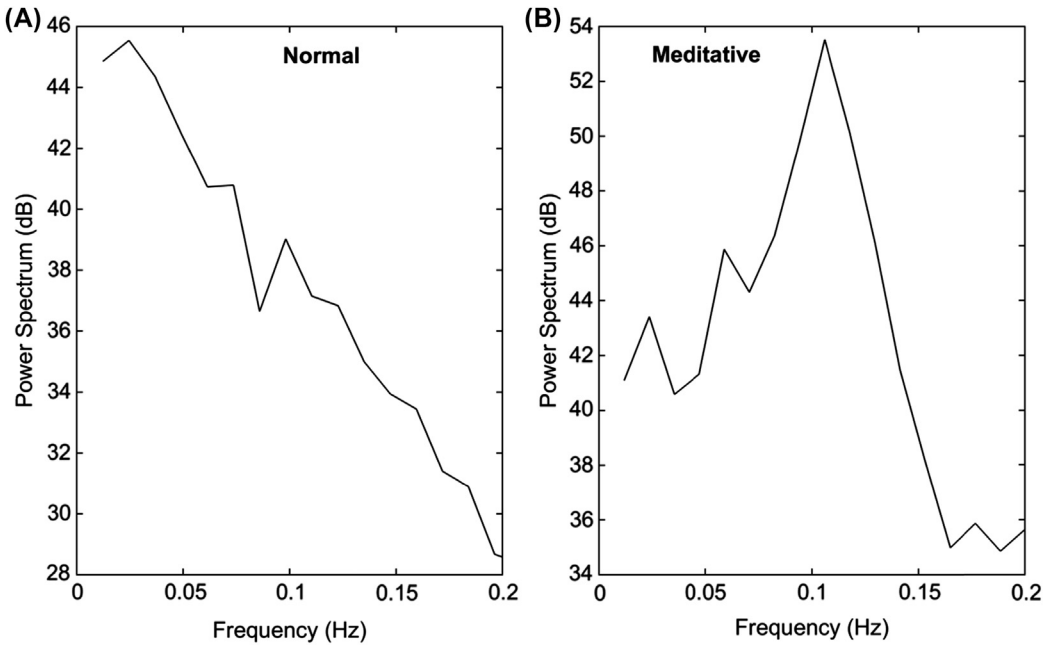

FIGURE 4.20 Normal and meditative spectra of heart rate variability data replotted in dB by taking 10 log of the power spectral curves in Figure 4.19. (A) The spectrum obtained under normal conditions is seen to decrease linearly with frequency. (B) The spectrum obtained under meditative conditions shows a large peak at 0.12 Hz. A reduced version of this peak is also seen in the normal data.

the word "bandwidth" relates to a signal's general ability to carry information. More specifically, we engineers define the term with regard to the range of frequencies found in a signal.

Figure 4.21A shows the spectrum of a hypothetical signal that contains equal energy at all frequencies up to 200 Hz, the frequency labeled f_c. Above this frequency, the signal contains no energy. We would say that this signal has a "flat" spectrum up to frequency f_c, and no energy above that frequency. That critical frequency, f_c, is called the "cutoff frequency." Since bandwidth is defined in terms of a signal's energy range and this signal contains energy only between 0 and f_c Hz, the bandwidth would be defined as from 0 to 200 Hz, or simply 200 Hz. The frequency range below 200 Hz is called the "passband," whereas the frequency range above 200 Hz is call the "stopband," as labeled in Figure 4.21A.

Although some real signals can be quite flat over selected frequency ranges, they are unlikely to show such an abrupt cessation of energy above a given frequency as seen in Figure 4.21A. Figure 4.21B shows the spectrum of a more realistic signal, where the energy decreases gradually. When the decrease in signal energy takes place gradually, as in Figure 4.21B, defining the signal bandwidth is problematic. If we want to attribute a single bandwidth value for this signal, we need to define a "cutoff frequency": a frequency defining a boundary between the region of substantial energy and the region of minimal energy. Such a boundary frequency has been arbitrarily defined as the frequency when the signal's value has declined by 3 dB with respect to its average unattenuated RMS value. (In Figure 4.21, the

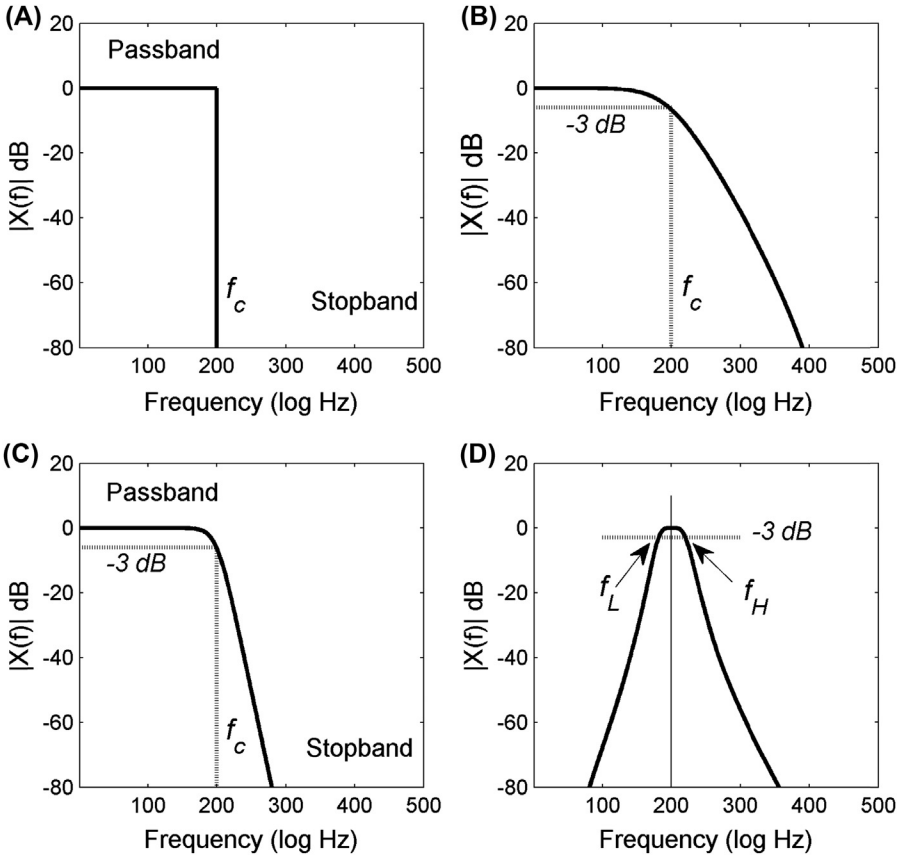

FIGURE 4.21 Frequency characteristics of ideal and realistic signals. The magnitude frequency plots shown here are plotted in dB. (A) A signal with an idealized, well-defined frequency range: 0–200 Hz. (B) A more realistic signal where signal energy decreases gradually at higher frequencies. (C) A realistic signal where signal energy decreases more sharply at higher frequencies. (D) A realistic signal where the signal energy decreases both above and below a "center frequency" of 200 Hz.

nominal unattenuated value for all signals is normalized 0 dB). The negative 3 dB boundary is not entirely arbitrary. If we convert −3 dB to linear units using Equation 2.9 we get: $10^{(-3\text{dB}/20)} = 0.707$. Thus the amplitude of a signal reduced by 3 dB is 0.707 of its unattenuated value. A signal's power is proportional to the amplitude squared (Equation 2.7), so at this attenuation, the signal's power is $0.707^2 = 0.5$. When the amplitude of the signal is attenuated by 3 dB from its nominal value, its amplitude is reduced by 0.707 and it power is halved. (Since the dB scale is logarithmic, −3 dB means a reduction of 3 dB; see Table 2.3.)

When the signal magnitude spectrum (the magnitude spectrum, not the power spectrum) is reduced by 3 dB, the power in the signal is half the nominal value, so this boundary frequency is also known as the "half-power point." The terms "−3 dB point," "3 dB down point," and "half-power point" are all used synonymously to define the boundary frequency,

f_c. In Figure 4.21B, the signal again has a bandwidth of 0.0−200 Hz, or just 200 Hz. The signal in Figure 4.21C has a sharper decline in energy, referred to as the "rolloff," but it still has a bandwidth of 200 Hz based on the −3 dB point.

It is possible that a signal "rolls off" or "attenuates" at both the low-frequency and high-frequency ends as shown in Figure 4.21D. In this case the signal has two cutoff frequencies, one labeled f_L and the other f_H. For such signals, the bandwidth is defined as the range between the two cutoff frequencies (or −3 dB points), that is, $BW = f_H - f_L$ Hz.

EXAMPLE 4.7

Find the effective bandwidth of the noisy signal, `x`, in file `Ex4_7_data.mat`. $f_s = 500$ Hz.

Solution: To find the bandwidth we first need the spectrum. Since the signal is noisy, we use spectral averaging to produce a cleaner spectrum. As explained in the results, test runs indicated that the Welch method using a segment length of 256 samples with maximal overlap works well. We then use MATLAB's `find` routine to search the magnitude spectrum for the first and last points that are greater than 0.5 (since we are using the power spectrum).

```
% Example 4.7 Find the effective bandwidth of the signal x in file Ex4_7_data.mat
%
load Ex4_7_data.mat;                    % Load the data file. Data in x
fs = 500;                               % Sampling frequency (given)
[PS1,f1]  =  welch(x,length(x),0,fs); % Determine  and  plot  the  un-averaged
spectrum
  ......plot, axes labels and new figure.......
%
nfft = 256;                             % Power spectrum window size
[PS,f] = welch(x,nfft,nfft-1,fs);       % Compute avg. spectrum, maximum overlap
PS = PS/max(PS);                        % Normalize peak spectrum to 1.0
plot(f,PS);  hold on;                   % Plot the normalized, average spectrum
.......axes labels.......
%
i_fl = find(PS > .5, 1, 'first');       % Find index of low freq. cutoff
i_fh = find(PS > .5, 1, 'last');        % Find index of high freq. cutoff
f_low = f(in_fl);                       % Convert low index to cutoff freq.
f_high = f(in_fh);                      % Convert high index to cutoff freq.
  .......plot markers.......
BW = f_high - f_low;                    % Calculate bandwidth
title(['Bandwidth: ',num2str(BW,3),' Hz']);  % Display bandwidth in title
```

Results: The unaveraged power spectrum shown in Figure 4.22 was determined using the `welch` routine, but with the segment size equal to the data length. Yes, we could have taken the square of the magnitude Fourier transform, but the `welch` routine provides the frequency vector, making life easier. The unaveraged power spectrum is quite noisy, as seen in Figure 4.22, so we do want to use spectral averaging. Using the `welch` routine with a window size of 256 samples produces a spectrum that is fairly smooth, but still has a reasonable spectral resolution, Figure 4.23. (The effect of changing window size is explored in one of the problems.)

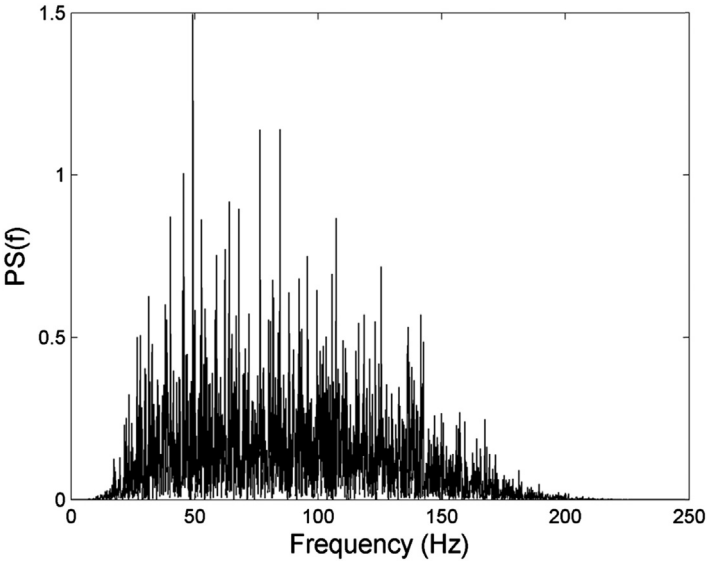

FIGURE 4.22 The power spectrum of the noisy signal used in Example 4.7. The difficulty of finding the general shape of this curve suggests that spectral averaging is in order. The averaged power spectrum is shown in Figure 4.23.

The `find` routine with the proper options gives us the indices of the first and last points above 0.5, and these are plotted superimposed on the spectral plot (large dots in Figure 4.23). These high and low sample points can be converted to frequencies using the frequency vector. The difference between the two cutoff frequencies is the bandwidth and is 111 Hz as displayed in the title of the

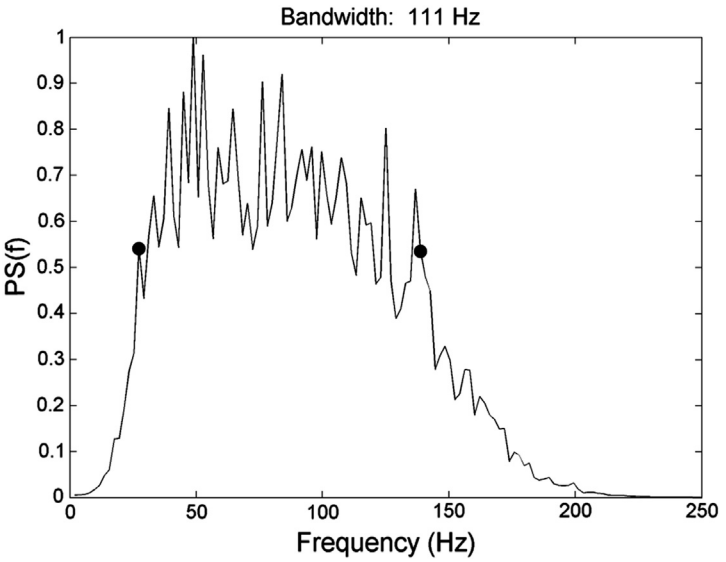

FIGURE 4.23 The averaged power spectrum of the signal used in Example 4.7. The window size is selected empirically to be 256 samples with maximal overlap. This size appears to give a smooth spectrum while maintaining good resolution. The cutoff frequency is found by searching for the half-power points. MATLAB's `find` routine is used to locate these points, which are identified as large dots on the spectrum. The frequency vector is used to convert the index of these points to equivalent frequencies and the calculated bandwidth is shown in the figure title.

spectral plot, in Figure 4.23. The estimation of the high and low cutoff frequencies could have been improved by interpolating between the spectral frequencies on either side of the 0.5 values. This is done in one of the problems.

4.6 TIME–FREQUENCY ANALYSIS

All Fourier analyses are done on a block of data, so the result represents a signal's spectrum over a finite period of time. An underlying and essential assumption is that the data have consistent statistical properties over the length of the data set being analyzed; that is, the signal is stationary over the analysis period (see Section 1.4.2.1). There is a formal mathematical definition of stationary based on consistency of the data's probability distribution function, but for our purposes, a stationary signal is one that does not change its mean, standard deviation, or autocorrelation function over time.

Most biological signals are not stationary, so to use Fourier analyses and many other signal processing techniques, some additional data transformation may be needed as mentioned in Chapter 1. Taking a page from spectral averaging described in Section 4.4, we could try to break up a signal into stationary segments and then perform a Fourier transform on each segment. This approach is known as "time–frequency" analysis; each segment produces a spectrum that covers a specific time period of the original signal. For example, if a 1-h data set was divided into sixty 1-min segments and a Fourier analysis performed on each segment, the resulting data would be 60 spectra, each describing a different time period.

There are many different approaches to performing time–frequency analyses, but the one based on the Fourier series is the best understood and the most popular. Since it is based on dividing a data set into shorter segments, sometimes quite short, it is called the "short-term Fourier transform" or "STFT." Since the data segments are usually short, a window function such as the Hamming window is usually applied to each segment before the Fourier transform is taken.

The STFT is very easy to implement in MATLAB: it is just like spectral averaging except, rather than averaged, each spectrum is plotted separately. Usually the resulting spectra are displayed on a three-dimensional plot with the spectra's magnitude on the vertical axis, and time and frequency on the other axes. The next example illustrates how easy it is to evaluate the STFT in MATLAB.

EXAMPLE 4.8

Load the respiratory signal found as variable `resp` in file `Resp_long.mat`. This signal is similar to the one used in Example 3.12 except that it is 1 h long. The sampling frequency is 125 Hz. Find the power spectra for sixty 1-min segments that overlap by 30 s (i.e., 50%). Plot these spectra in three dimensions using MATLAB's `mesh` routine. Limit the frequency range to 2 HZ and do not plot the DC components.

Solution. Simply modify the `welch` routine so that it saves the individual spectra in a matrix rather than computing an average spectrum. Also add code to compute the relative time corresponding to each segment. The main program is given below followed by the modified `welch` routine now called `stft` for the STFT.

```
% Example 4.8 Example of the STFT time–frequency analysis.
%
load Resp;                          % Get respiratory data
N = length(resp);                   % Determine data length
fs = 125;                           % Sampling frequency
nfft = fs*60;                       % 1 min of samples
noverlap = round(nfft/2);           % Use 50% overlap
m_plot = round(2/(fs/nfft));        % Find m for 2 Hz
[PS,f,t] = stft(resp',nfft,noverlap,fs);     % Calculate the STFT
PS1 = PS(:,2:m_plot);               % Resize power spectra to range to 2 Hz
f1 = f(1:m_plot-1);                 % Resize frequency vector for plotting
mesh(f1,t,PS1);                     % Plot in 3D
view([17 30]);                      % Adjust view for best perspective
  ......labels.......
```

The modified `welch` routine is:

```
function [PS,f,t] = stft(x,nfft,noverlap,fs);
% Function to calculate short term Fourier Transform
%
  .....same as welch.m up to the calculation of the power spectra.......
% Data have been windowed
  PS_temp = abs((fft(data)).^2)/N1;     % Calculate PS (normalized)
  PS(k,:) = PS_temp(2:half_segment+1)/(nfft/2);   % Remove redundant points
  t(k) = (k - 1)*nfft/fs;               % Construct time vector
end                                     % Program ends here
```

Results: The `stft` routine generates a matrix, `PS`, that contains the spectra in rows. The frequency vector, `f`, is constructed as in `welch` and not shown here. There is also a time vector, `t`, that shows the time when each segment starts.

The results from the STFT are shown in Figure 4.24. The view was initially adjusted interactively to find a perspective that showed the progression of power spectra. In Figure 4.24, each time frame shows a spectrum that is similar to the spectrum in Figure 3.24, with a large peak around the respiratory rate.

4.7 SUMMARY

Converting a real-world continuous signal to a digital signal requires three operations: time sampling, amplitude slicing, and, usually, truncation. The DFT can be used to understand the relationship between a continuous time signal and the sampled version of that signal. This frequency-based analysis shows that the original, unsampled signal can be recovered if the sampling frequency is more than twice the highest frequency component in the unsampled signal. Quantization is viewed as noise added to the signal and can be quantitatively determined from the amplitude and quantization level (i.e., number of bits) of the

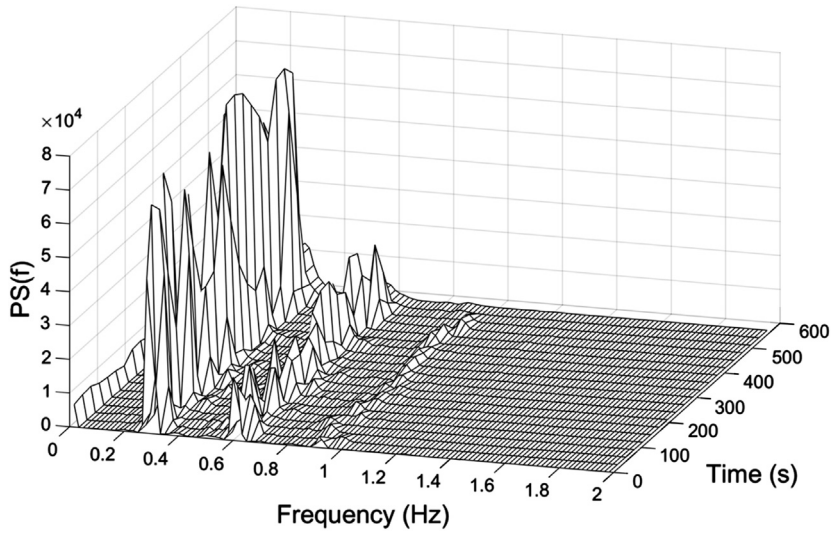

FIGURE 4.24 Time–frequency plot of a 1-h respiratory signal. This plot was constructed by segmenting the 1-h signal into 60-sec increments and taking the power spectrum of each increment separately rather than averaging as in the Welch method. Segments overlap by 50%.

ADC. Data truncation can produce discontinuities at the endpoints that may result in artifacts in the DFT, particularly for short data segments. Tapering window functions can be imposed on the digitized data to reduce these artifacts, but for reasonable signal lengths (>256 samples), they are not needed.

The Fourier transform can be used to construct the power spectrum of a signal by taking the square of the magnitude spectrum. The power spectral curve shows signal power as a function of frequency. The power spectrum is particularly useful for noisy or random data where phase characteristics have little meaning. By dividing the signal into a number of possibly overlapping segments and averaging the power spectrum obtained from each segment, a smoothed power spectrum can be obtained.[7] The resulting averaged power spectrum curve will emphasize the broadband or general characteristics of a signal's spectrum, but at the cost of fine detail.

Bandwidth is a term used commonly to describe the capability of a communication channel to carry information. Bandwidth is specifically defined as the range of frequencies included in a signal. To be included, the energy associated with a given frequency must be greater than half that of the signal's nominal values. So for frequencies to be included, the signal amplitude must be greater that 0.707 of the nominal value or, equivalently, attenuated no more than −3 dB from these values.

Nonstationarity is a common problem in biological signals. For spectral analysis, the STFT is a common approach to dealing with such signals. The idea is to divide the signal into short

[7]Although averaging is generally applied to the power spectrum, the magnitude spectrum can also be used. Averaging cannot be applied to the phase spectrum because the phase is altered by the time shifting inherent in averaging.

segments that are, we hope, stationary, and then apply the Fourier transform to each separate segment. This results in a number of spectra that represent a specific time frame of the overall signal, hence the term "time–frequency analysis." These spectra are usually displayed on three-dimensional plot of time, frequency, and spectral magnitude (or power spectrum). Alternatively, time–frequency information could be represented in a two-dimensional plot of time and frequency where color or shading maps the magnitude value. In principal, phase information could also be displayed this way, but the sensitivity of phase to time shifts would make such displays difficult to interpret.

PROBLEMS

1. This problem demonstrates aliasing similar to Example 4.1. Generate a 512-point waveform consisting of two sinusoids at 200 and 400 Hz. Assume a sampling frequency of 1 kHz. Generate another waveform containing frequencies at 200 and 900 Hz. Take the Fourier transform of both waveforms and plot the magnitude of the spectrum up to $f_s/2$. Plot the two spectra superimposed, but in different colors to highlight the additional peak due to aliasing at 100 Hz. [Hint: Follow the approach in Example 4.1 to generate the sine waves.]
2. Load the chirp signal, `x`, in file `chirp.mat` and plot the magnitude spectrum. The sample frequency is 5 kHz. Now decrease the sampling frequency by a factor of two by 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 half that of the original signal.) Decreasing the sampling frequency of an existing signal is termed "downsampling" and can be done easily in MATLAB. To downsample by a factor of two use: `x1 = x(1:2:end)`.
3. Load the file `sample_rate.mat`, which contains signals `x` and `y` sampled at 1 kHz. Is either of these signals likely to be undersampled (i.e., $f_s/2 \leq f_{max}$)? Alternatively, could the sampling rate of either signal be safely reduced? Justify your answer.
4. Repeat Problem 3 for the data in `sample_rate1.mat`, which contains signals `x` and `y`, also sampled at 1 kHz. Is either of these signals undersampled or could the sampling rate of either signal be safely reduced? Again, justify your answer.
5. Generate a 1024-point waveform consisting of four sinusoids at 100, 200, 300, and 400 Hz. Make $f_s = 1$ kHz. Generate another waveform containing frequencies at 600, 700, 800, and 900 Hz. Take the Fourier transform of both waveforms and plot the magnitude of the full magnitude spectrum (i.e., 1024 points) on separate plots. (Plotting one above the other using `subplot` makes a nice comparison.)
6. The file labeled `quantization.mat` contains three vector variables: `x`, `y`, and `z`, all sampled at 1 kHz. This file also contains a time vector, `t`, useful for plotting. Variable `x` represents an original signal. Variable `y` is the same signal after it has been sampled by a 6-bit ADC, and variable `z` is the original signal after sampling by a 4-bit ADC. Both converted signals have been scaled to have the same range as `x`, 0–5 volts. On one plot, show the three variables superimposed and on another plot, show the error

signals `x - y` and `x - z`. Then calculate the RMS error between `x` and `y` and `x` and `z`. Also calculate the theoretical error for the two ADCs based on Equation 4.3.

7. The file `short.mat` contains a very short signal of 32 samples, $f_s = 40$ Hz. Plot the nonredundant magnitude spectrum as discrete points obtained with and without zero padding. Zero pad out to a total of 256 points.
8. Generate a short signal consisting of a short impulse function: `x = [1 0 0 0 0 0];`, ($f_s = 50$ Hz). As in Problem 7, find the magnitude transform with no padding and zero padding to 256 points. Plot the nonredundant spectrum.

   Repeat for a double impulse function of the same length: `x = [1 0 0 1 0 0];`. Note how information about the impulse function is not really improved by padding, but really makes a difference for the double impulse.
9. The variable `x` in file prob4_9_data contains 200 and 300 Hz sine waves with SNRs of −6 dB. The sampling frequency is 1000 Hz and the data segment is fairly short; 64 samples. Plot the magnitude spectrum obtained with and without a Hamming window. Use Equation 4.9 to generate a window function 64 points long and multiply point by point with the signal variable `x`. Note that the difference is slight, but could be significant in certain situations.
10. 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);`). (Recall that `sig_noise` assumes $f_s = 1$ kHz.) Calculate and plot the power spectrum using two different approaches. In the first approach, use the direct approach: take the Fourier transform and square the magnitude function. In the second approach, use the traditional method defined by Equation 4.11: take the Fourier transform of the autocorrelation function. Calculate the autocorrelation function using cross`corr(x,x)`, 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 power spectrum derived from both techniques. The scales will be different because the MATLAB `fft` routine does not normalize the output. You may find a very slight difference between results because of the difference in computer round-off errors.
11. Construct two arrays of white noise using `randn`: one 128 points in length and the other 1024 points in length. ($f_s = 1$ kHz.) Take the power spectrum of both. Eliminate the first point—the average or DC term—when you plot the spectra and plot only nonredundant points. Does increasing the length improve the spectral estimate of white noise, which should be flat? This demonstrates the benefit of spectral averaging when examining broadband features.
12. Use MATLAB routine `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. The MATLAB call should be:

    `x = sig_noise([320 340],-12,N);` where `N` = either 128 or 512.

    Calculate and plot the (unaveraged) power spectrum. Repeat the execution of the program several times and note the variability of the results indicating that noise is noisy. (Remember `sig_noise` assumes $f_s = 1$ kHz.)
13. Use `sig_noise` to generate a 128-point array containing 320 and 340 Hz sinusoids as in Problem 12. Calculate and plot the unaveraged power spectrum of this signal for an

SNR of −10, −14, and −16 dB. Plot all three on the same graph using subplot and execute the program several times and observe that variability. How does the presence of noise affect the ability to detect and distinguish between the two sinusoids? (Remember $f_s = 1$ kHz.)

14. Load the file `broadband2`, which contains variable `x`, a broadband signal with added noise. ($f_s = 1$ kHz.) Calculate the averaged power spectrum using the `welch` routine. 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, `x`. Use the default overlap.

15. Load the file `eeg_data.mat` that contains EEG data. ($f_s = 50$ Hz.) Analyze these data using the unaveraged power spectral technique and an averaging technique using the `welch` routine. Find a segment length that smooths the background spectrum, but still retains any important spectral peaks. Use a 99% overlap.

16. Modify the `welch` routine to create a routine `welch_win` that applies a "Blackman window" to the data before taking the Fourier transform. The Blackman window is another of many window functions. It adds another cosine term to the equation of the Hamming window giving:

$$w[n] = 0.41 + 0.5\cos\left(\frac{2\pi n}{N}\right) + 0.08\cos\left(\frac{4\pi n}{N}\right)$$

Load the file `broadband3.mat`, which contains a broadband signal and a sinusoid at 400 Hz in variable `x` ($f_s = 1$ kHz). Analyze the broadband/narrowband signal using both `welch` and `welch_win` with segment lengths of $N/4$ and $N/16$, where N is the length of the data of variable, `x`. Use the default overlap. Note that for the shorter segment ($N/16$), the Blackman window provides slightly more smoothing that the rectangular window used by `welch`.

17. Load the file broadband3.mat, which contains a broadband signal and a sinusoid at 400 Hz in variable x ($f_s = 1$ kHz). Analyze the broadband/narrowband signal using `welch` with two different overlaps: 50% (the default) and maximum. As in Problem 16, use segment lengths of $N/4$ and $N/16$, where N is the length of the data of variable, `x`.

18. Find the effective bandwidth of the signal `x` in file `broadband2.mat` ($f_s = 1$ kHz). Load the file `broadband2.mat` and find the bandwidth using the methods of Example 4.7. Find the power spectrum using `welch` and a window size ranging between 50 to 200 points and the maximum overlap. Find the window size that appears to give the best estimate of bandwidth. (Note: window sizes that produce the best looking power spectrum may not result in the most accurate estimate of bandwidth. Can you explain?)

19. Load the file `broadband2.mat` and find the bandwidth using the methods of Example 4.7, but this time using interpolation ($f_s = 1$ kHz). Determine the power spectrum using `welch` with a window size of 100 and the maximum overlap. This will give a highly smoothed estimate of the power spectrum but a poor estimate of bandwidth because of the small number of points. Interpolation can be used to improve the bandwidth estimate from the smoothed power spectrum. The easiest way to implement interpolation in this problem is to increase the number of points in the power spectrum using MATLAB's `interp` routine. (For example, `PS1 = interp(PS,5)` increases the number of points in PS by a factor of 5.) Estimate the bandwidth using

the approach of Example 4.7 before and after expanding the power spectrum by a factor of five. Be sure to expand the frequency vector (`f` in Example 4.7) by the same amount.

20. Load the file `ECG_1hr`, which contains 1 h of electrocardiogram signal in variable `ecg.mat`. Plot the time–frequency spectra of this signal using the STFT approach shown in Example 4.8 (`stft.m` can be found on the associated files). Limit the frequency range to 3 Hz and do not include the DC term. The sample frequency is 250 Hz. [Hint: You can improve the appearance of the higher frequency components by limiting the z axis using `ylim` and the color axis using `caxis`.]

21. Repeat Problem 20, but change the parameters to emphasize a different feature of this signal: the influence of respiration and other factors on heat rate. Make the segment length 30 s and the overlap 20 s (2/3). Also change the upper limit on the frequency range to 8 Hz, but again do not include the DC term ($f_s = 250$ Hz).

Plot the resulting spectra as a three-dimensional "heat map"-type display. Use `pcolor` with shading set to `interp` to plot the spectra and correctly label and scale the two axes. Limit the color range to 200. You will see a light-colored wavy line around the heart rate frequency (approximately 1 Hz) that reflects the variation in heart rate due to respiration. Now lower the upper limit of the color range to around 20 and note the chaotic nature of the higher frequency components. [Hint: If you make the lower limit of the color range slightly negative, it lightens the background and improves the image.]