CHAPTER

2

Signal Analysis in the Time Domain

2.1 GOALS OF THIS CHAPTER

Signals are the foundation of information processing, transmission, and storage. Signals also are the basis for communication between biological processes, Figure 2.1, and for our interaction with those processes. If we are going to work with them, we need to understand them.

To recap some basic concepts from Chapter 1, signals differ in the way they are represented and they also exhibit different properties. Signal representations are unique; a signal

FIGURE 2.1 Signals course relentlessly through the body. As with all signals, they must be carried by some form of energy. Biosignals are carried by electrical energy in the nervous system or by chemical energy as molecular signatures in the endocrine system and other biosystems. Measurement and analysis of biosignals is fundamental to diagnostic medicine and biomedical research.

TABLE 2.1 Major Signal Properties

Signal Properties	
Linear	Nonlinear
Deterministic	Stochastic (random)
Stationary (time invariant)	Nonstationary

is either analog or digital, time domain or frequency domain. Signal representations are summarized in Table 1.2. In this chapter, we work only with the time domain representations of signals.

Signal properties are less definitive. A signal can contain a mixture of properties; for example, a signal can be considered deterministic yet still contain some random behavior.[1] The basic signal properties introduced in Chapter 1 are summarized in Table 2.1.

In this chapter we will:

- Master some very basic, and not so basic, measurements applied to the time domain representations of signal and noise.
- Define measurements that quantify the strength of the signal and that quantify the ratio of signal to noise in a waveform.
- Define measurements that define a signal's stochastic properties (i.e., its variability or randomness).
- Describe a powerful method for reducing the influence of noise under certain circumstances.
- Define and work with the most important waveform of all, the sinusoid.
- Introduce some more involved time domain analytic techniques based on the concept of correlation that quantifies the similarity between two signals or functions.
- Apply these correlations techniques to shifted versions of two signals, or between different segments of the same signal.

In Chapter 3, we use correlations between signals and some carefully selected sinusoids to transform signals to the frequency domain. Signal comparisons using correlation are very useful and, since we do it in MATLAB, not all that difficult.

2.2 TIME DOMAIN MEASUREMENTS

Sometimes all we need to do is look at a signal and we can get the information we need. For example, a radiologist can often make a diagnosis by just looking at an image. Similarly a cardiologist can sometimes make diagnoses after a quick, qualitative examination of an electrocardiography signal. But most of the time we must probe deeper. We may need to get

[1]In a strict sense, any signal that contains randomness is a stochastic signal, but if the randomness is small enough that it can be ignored, we could consider it deterministic.

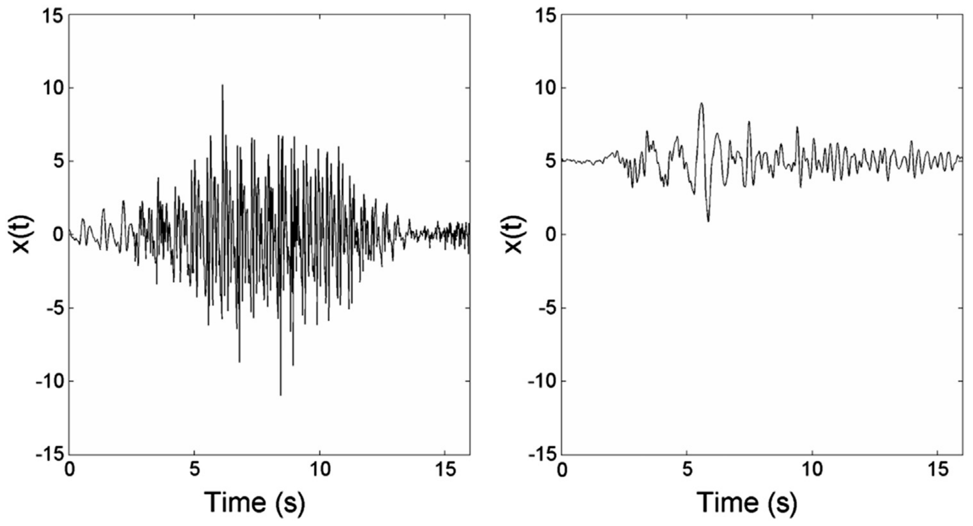


FIGURE 2.2 Two clearly different electroencephalogram signals. The one on the right has higher overall values because it has a higher mean level, but the signal on the left appears more energetic because it has more fluctuation.

quantitative measurements from a signal, some numbers that summarize its medically useful features. We begin with a discussion of the two most basic measurements you can make on a signal: its average value and its effective strength.

2.2.1 Mean and Root Mean Squared Signal Measurements

Figure 2.2 shows two electroencephalogram (EEG) signals that are obviously different. The signal on the right has higher overall values, but because it has less fluctuation it appears less energetic than the one on the left. The two most basic signal measurements, the mean and root mean squared (rms) value, quantify these two differences. The equations we use to calculate these measurements depend on whether the domain used to represent the signal is discrete or continuous. Although the same measurement requires a different equation for discrete versus continuous signals, the two equations are related.

For a discrete signal that is represented by a series of numbers, determination of mean value is intuitive; it is the average of all the numbers in the series. To determine the average of a series of numbers, simply add the numbers together and divide by the length of the series (i.e., the number of numbers in the series). For a series of N numbers:

$$x_{avg} = \overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{2.1}$$

where n is an integer indicating a specific member in the series and ranging from 1 to N.[2] Some equations show summation, or similar operations, as ranging between 0 and $N - 1$.

[2]In this text, the letters n, k, and i are used as an index to identify a specific member in a series of numbers.

However, in MATLAB the index, n, must be nonzero, so for compatibility we usually use a range of 1 to N. The bar over the x in Equation 2.1 stands for "the average of" whatever variable it is over, in this case the average of x.

For signals represented in the continuous domain, summation becomes integration. The discrete set of numbers, x_n, becomes a continuous time series, $x(t)$, and the length of the series, N, becomes a fixed time period, T. This leads to the equation:

$$\overline{x} = \frac{1}{T}\int_0^T x(t)dt \tag{2.2}$$

In general, although different equations are required for the same operation in the discrete and continuous domains, these equations are related. Specifically, summation in the discrete domain becomes integration in the continuous domain, discrete variables become continuous variables, and the number of points used to represent a discrete signal becomes a fixed time period. Moreover, as defined in Chapter 1, the variable index is related to time, $n = t f_s$, and the total number of points is related to the total time, $N = T f_s$, where f_s is the sampling frequency. These equivalences are shown in Table 2.2 and provide a simple algorithm for converting from discrete domain equations to time domain equations and vice versa. Whether signals in the two domains contain the same information is another matter that is covered in Chapter 4.

The other basic difference between the two signals in Figure 2.2 is their range of variability. The signal on the left clearly shows greater and more prevalent fluctuations. This signal property is quantified by the rms value. The equation for the rms of a signal follows this measurement's name: first square the signal, then take its mean, then take the square root of this mean:

$$x_{rms} = \left[\frac{1}{N}\sum_{n=1}^{N} x_n^2\right]^{1/2} \tag{2.3}$$

TABLE 2.2 Relationship Between Discrete and Continuous Operations

Operation	Discrete	Continuous
Summation/integration	$\sum_{n=1}^{N} x_n$	$\int_{t=0}^{T} x(t)$
Variable name	x_n	$x(t)$
	$n = t f_s$	$t = n/f_s$
Signal length	N	T
	$N = T f_s$	$T = N/f_s$

The continuous version of the equation is obtained by following the simple rules in Table 2.2.

$$\bar{x}_{rms} = \left[\frac{1}{T}\int_0^T x(t)^2 dt\right]^{1/2} \tag{2.4}$$

Note that the discrete equation (Equation 2.3) is easy to implement on a computer (even easier in MATLAB as shown later), but the continuous equation (Equation 2.4) needs to be solved "analytically," or by hand with paper and pencil. The next example finds the rms value of a simple signal, a sine wave, in both the continuous and discrete domains.

EXAMPLE 2.1

Find the rms value in both the continuous and discrete domains of the sinusoidal signal: $x(t) = A\sin\left(\frac{2\pi t}{T}\right) = A\sin(2\pi f t)$.

Analytical Solution: Since this signal is periodic, with each period being the same as the previous one, it is sufficient to apply the rms equation over a single period. (This is true for most operations on sinusoids.) Applying Equation 2.4 along with a little calculus including a good table of integrals:

$$\begin{aligned}\bar{x}(t)_{rms} &= \left[\frac{1}{T}\int_0^T x(t)^2 dt\right]^{1/2} = \left[\frac{1}{T}\int_0^T \left(A\sin\left(\frac{2\pi t}{T}\right)\right)^2 dt\right]^{1/2}\\ &= \left[\frac{1}{T}\frac{A^2T}{2\pi}\left(-\cos\left(\frac{2\pi t}{T}\right)\sin\left(\frac{2\pi t}{T}\right) + \frac{\pi t}{T}\right)\Bigg|_0^T\right]^{1/2}\\ &= \left[\frac{A^2}{2\pi}(-\cos(2\pi)\sin(2\pi) + \pi + \cos 0 \sin 0)\right]^{1/2}\\ &= \left[\frac{A^2\pi}{2\pi}\right]^{1/2} = \left[\frac{A^2}{2}\right]^{1/2} = \frac{A}{\sqrt{2}} = 0.707A\end{aligned}$$

Hence, there is a proportional relationship between the "peak-to-peak" amplitude of a sinusoid ($2A$ in this example) and its rms value: the rms value is $1/\sqrt{2}$ (rounded here to 0.707) times the amplitude, A. This is only true for sinusoids. For other waveforms, you need to apply the defining equation (Equation 2.4).

Discrete Solution: Here we take advantage of a useful MATLAB routine that calculates the mean (i.e., average) of an array.

```
xm = mean(x);     % Evaluate mean of x
```

where x is an array that in our case will contain the sine wave signal. (Note that if x is a matrix, the output is a row vector that is the mean of each column of the matrix. We use this powerful feature of this routine in a later example.)

To solve the problem on a computer, we first need to generate the sine wave and we need definitive values for f (or T) and A. One of the few downsides of computer solutions is that we cannot solve a problem with general variables. Arbitrarily we assign $f = 4$ Hz and $A = 1.0$. Also arbitrarily, we choose an N of 500 and a sampling frequency of 500 Hz. We then follow the approach used in Example 1.6, generate a time array, and use that to produce the sine wave

```
N = 500;                % Number of points (arbitrary)
fs = 500;               % Sampling frequency in Hz (arbitrary)
f = 4;                  % Sine wave frequency in Hz (arbitrary)
t = (0:N-1)/fs;         % Generate time array N points long (or: t = (0:N)/fs;)
x = sin(2*pi*f*t);      % Generate sine wave
```

Next, perform the rms operation using the `sqrt` and `mean` routines.

```
rms = sqrt(mean(x.^2)); % Calculate rms
disp(rms)               % and display
```

Note that the .^ operator must be used for squaring `x` since we want each point squared separately. The program then becomes:

```
% Example 2.1 find the rms value of a discrete sine wave.
%
fs = 500;                  % Sample interval
N = 500;                   % Number of points
f = 4;                     % Frequency of sine wave
t = (1:N)/fs;              % Generate the time vector
x = sin(2*pi*f*t);         % Generate sine wave
rms = sqrt(mean(x.^2));    % Calculate rms
disp (rms)                 % and display
```

Result: The calculated rms value determined by this program was 0.7071, which is the same as that found analytically if the amplitude, A, of the sine wave equals 1.0. Although the MATLAB approach is easier for most of us, it does not provide the general solution of the analytical approach.

2.2.1.1 Decibels

It is common to compare the intensity of two signals using ratios (i.e., V_{Sig1}/V_{Sig2}), particularly if the two are the signal and noise components of a waveform. Moreover, these signal ratios are commonly represented in units of "decibels" or "dB." When decibels are applied to ratios, they are dimensionless since whatever units are involved cancel. So decibels are not really units, but a logarithmic scaling of dimensionless ratios. The decibel has several features: (1) it provides a measurement of the effective power, or rather power ratio; (2) the log operation compresses the range of values (for example, a range of 1–1000 becomes a range of 1−3 in log units); (3) when numbers or ratios are to be multiplied, they are simply added if they are in log units; and (4) the logarithmic characteristic is similar to human perception. It is this latter feature that motivated Alexander Graham Bell to develop the logarithmic unit called the "Bel." Audio power increments in logarithmic Bels were perceived as equal increments by the human ear. The Bel turned out to be inconveniently large, so it has been replaced by the decibel: dB = 1/10 Bel.

Although the dB unit was originally defined only in terms of a ratio, the convenience of dB units has motivated their use in applications that do not involve ratios. In such cases, the dB actually has a dimension, the dimension of the signal (dB volts, dB dynes, etc), but these units are often not specifically stated. So when you see dB, you need to look at the context to see if it has dimensions.

When applied to a power measurement, the decibel is defined as 10 times the log of the power ratio:

$$P_{dB} = 10\log\left(\frac{P_2}{P_1}\right)\text{dB} \tag{2.5}$$

As described in the next paragraph, the power of a signal is proportional to the amplitude (in rms) squared. So when it is applied to the voltage ratio, the decibel is defined as 20 times the log of the voltage ratio since $10\log x^2 = 20\log x$. The same is true when it is applied to just a voltage (i.e., not a ratio). So the definition of dB when applied to voltage (or other nonpower units) is:

$$\begin{aligned} v_{dB} &= 10\log\left(\frac{v_2^2}{v_1^2}\right) = 20\log\left(\frac{v_2}{v_1}\right) \quad \text{or} \\ v_{dB} &= 10\log\left(v_{rms}^2\right) = 20\log\left(v_{rms}\right) \end{aligned} \tag{2.6}$$

To demonstrate that power is proportional to voltage squared, consider the power that can be produced by voltage v. To produce energy, it is necessary to feed the voltage into a resistor, or a resistor-like element, that consumes energy. (Recall from basic physics that resistors convert electrical energy into thermal energy, i.e., heat.) The power (energy per unit time) induced in the resister from the voltage is given by:

$$P = \frac{v_{rms}^2}{R} \tag{2.7}$$

where R is the resistance. Equation 2.7 shows that the power transferred to a resistor by a given voltage depends, in part, on the value of the resistor. Assuming a nominal resistor value of 1 Ω, the power will be equal to the voltage squared; however, for any resistor value, the power transferred will still be proportional to the voltage squared. When dB units are used to describe a ratio of voltages, the value of the resistor is irrelevant, since the resistor values will cancel out, assuming that the two values of R represent the same resistor:

$$v_{dB} = 10\log\left(\frac{v_2^2/R}{v_1^2/R}\right) = 10\log\left(\frac{v_2^2}{v_1^2}\right) = 20\log\left(\frac{v_2}{v_1}\right) \tag{2.8}$$

If dB units are used to express the intensity of a single signal, then the units will be proportional to the log power in the signal.

To convert a voltage from dB to rms, use the inverse of the defining equation (Equation 2.8):

$$v_{rms} = 10^{v_{dB}/20} \tag{2.9}$$

Putting units in dB is particularly useful when comparing ratios of signal and noise to determine the signal-to-noise ratio as shown in the next section.

EXAMPLE 2.2

A sinusoidal signal is fed into an *attenuator* that reduces the intensity of the signal. The input signal has a peak amplitude of 2.8 V and the output signal is measured at 2 V peak amplitude. Find the ratio of output to input voltage in dB. Compare the power-generating capabilities of the two signals in linear units.

Solution: Convert each peak voltage to rms, then apply Equation 2.8 to the given ratio. Also calculate the ratio without taking the log.

$$V_{rms\ dB} = 20\log\left(\frac{V_{out\ rms}}{V_{in\ rms}}\right) = 20\log\left(\frac{2.0\times 0.707}{2.8\times 0.707}\right)$$
$$V_{rms\ dB} = -3\ \text{dB}$$

The power ratio is:

$$\text{Power ratio} = \frac{V^2_{out\ rms}}{V^2_{in\ rms}} = \frac{(2\times 0.707)^2}{(2.8\times 0.707)^2} = 0.5$$

Analysis: The ratio of the amplitude of a signal coming out of a process to that going into the process is known as the *gain*, and is often expressed in dB. When the gain is <1, it means there is actually a loss, or reduction, in signal amplitude. In this case, the signal loss is 3 dB, so the gain of the attenuator is actually −3 dB. To confuse yourself further, you can reverse the logic and say that the attenuator has an attenuation (i.e., loss) of +3 dB. In this example, the power ratio is 0.5, meaning that the signal coming out of the attenuator has half the power-generating capabilities of the signal that goes in. An attenuation of 3 dB is equivalent to a loss of half the signal's energy. Of course, it is not really necessary to convert the peak voltages to rms since a ratio of these voltages is taken and the conversion factor (0.707) cancels out.

2.2.1.2 *Signal-to-Noise Ratio*

Most waveforms consist of signal plus noise mixed together. As noted previously, signal and noise are relative terms, relative to the task at hand: the signal is what you want from the waveform, whereas the noise is everything else. Often the goal of signal processing is to separate a signal from noise, or to identify the presence of a signal buried in noise, or to detect features of a signal buried in noise.

The “signal-to-noise ratio” or “SNR” quantifies the relative amount of signal and noise present in a waveform. As the name implies, this is simply the ratio of signal to noise, both measured in rms amplitude, and is often expressed in dB:

$$\mathrm{SNR} = 20\log\left(\frac{\text{signal}}{\text{noise}}\right)\mathrm{dB} \tag{2.10}$$

To convert from dB scale to a linear scale:

$$\mathrm{SNR}_{\mathrm{Linear}} = 10^{\mathrm{dB}/20} \tag{2.11}$$

When expressed in dB, a positive number means the signal is greater than the noise and a negative number means the noise is greater than the signal. Some typical dB ratios and their corresponding linear values are given in Table 2.3.

To give a feel of what some SNR dB values actually mean, Figure 2.3 shows a sinusoidal signal with various amounts of white noise. When the SNR is +10 dB, the sine wave can be readily identified. This is also true when the SNR is +3 dB. When the SNR is −3 dB you can still spot the signal, at least if you know what to look for. When the SNR decreases to −10 dB, it is impossible to determine if a sine wave is present visually.

2.2.2 Variance and Standard Deviation

Signal properties such as the mean and rms apply to both deterministic and stochastic signals, but some signal properties are dedicated to describing a signal's random behavior. The most common description of signal randomness is the “sample variance,” s^2 (also commonly written as σ^2). The variance is a measure of signal fluctuation or variability and is insensitive to the signal's average value. The calculation of variance for both continuous and discrete signals is given as:

$$s^2 = \frac{1}{T}\int_0^T (x(t) - \overline{x})^2 dt \tag{2.12}$$

TABLE 2.3 Equivalence Between dB and Linear Signal-to-Noise Ratios

SNR in dB	SNR Linear
+20	$v_{\text{signal}} = 10\ v_{\text{noise}}$
+3	$v_{\text{signal}} = 1.414\ v_{\text{noise}}$
0	$v_{\text{signal}} = v_{\text{noise}}$
−3	$v_{\text{signal}} = 0.707\ v_{\text{noise}}$
−20	$v_{\text{signal}} = 0.1\ v_{\text{noise}}$

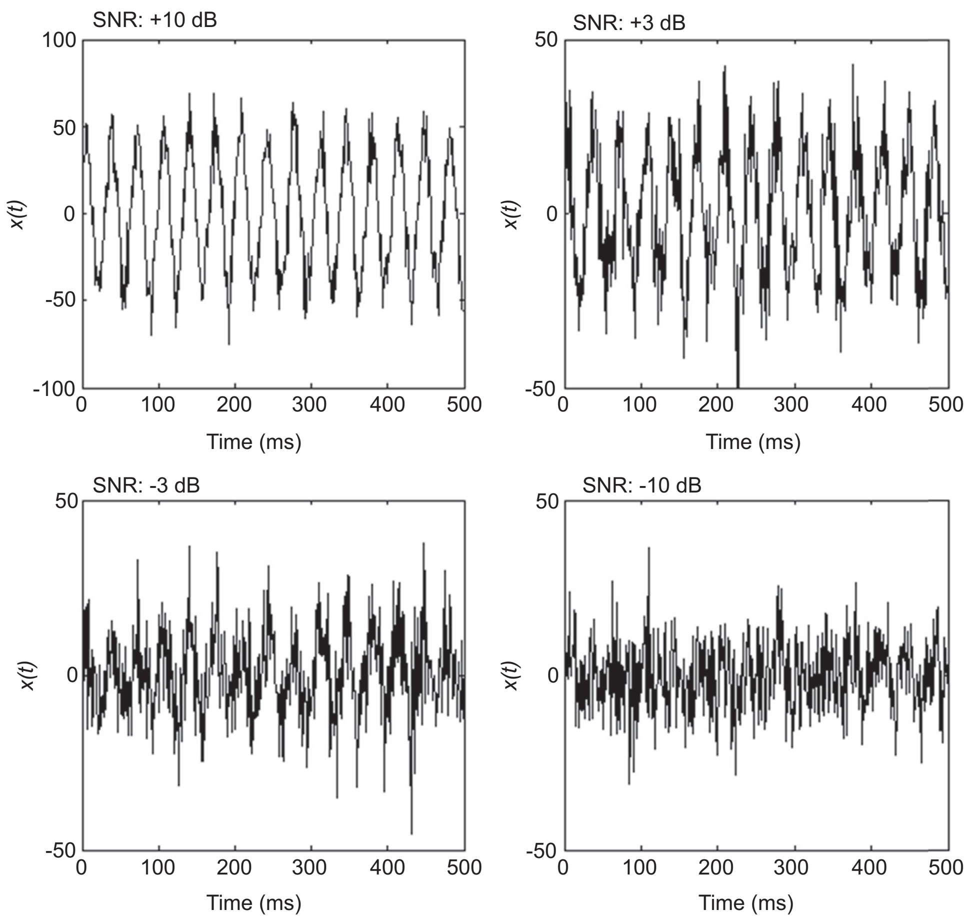


FIGURE 2.3 A 30 Hz sine wave with varying amounts of added noise. The sine wave is barely discernable when the SNR is −3 dB and is not visible when the SNR is −10 dB.

$$s^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x(t)-\overline{x})^2 \tag{2.13}$$

where $\overline{x}$ is the signal mean (Equation 2.1). Comparing Equation 2.13[3] with Equation 2.3, we see that if the signal mean is zero (i.e., $\overline{x} = 0$), then the variance is almost the same as the rms value. The difference is that in rms you divide the summation by the signal length $\left(\frac{1}{N}\right)$, whereas in the variance calculation you divide the summation by the signal length minus 1, i.e., $\left(\frac{1}{N-1}\right)$. The reason for this subtle difference, which becomes ever smaller as N increases, has to do with the definition of variance. Variance is defined in terms of the probability

[3]Some statisticians prefer to use the symbol s^2_{N-1} to indicate that the summation is normalized by $\frac{1}{N-1}$.

distribution of a random variable, which is generally unknown in real-world situations. So the variance measurements produced by Equations 2.12 and 2.13 are estimations of the true variance that are obtained from a limited sample. They also include estimations of the sample mean, $\overline{x}$, in those equations. Dividing by $N - 1$ reduces the bias produced by approximating the mean and results in what is known as the "unbiased sample variance." This is the equation most commonly used to estimate the variance from real data samples. Despite the similarities between the calculations of rms (for zero mean signals) and variance, they come from very different traditions (statistics versus measurement) and are used to describe conceptually different aspects of a signal: variance describes signal variability and rms describes signal magnitude.

The "standard deviation" is another measure of a signal's variability and is simply the square root of the variance:

$$s = \left[\frac{1}{T}\int_0^T (x(t) - \overline{x})^2 dt\right]^{\frac{1}{2}} \tag{2.14}$$

$$s = \left[\frac{1}{N-1}\sum_{n=1}^{N} (x_n - \overline{x})^2\right]^{\frac{1}{2}} \tag{2.15}$$

Calculating the variance or standard deviation of a signal using MATLAB is as easy as calculating the mean.

```
xv = var(x);       % Evaluate variance of x

xsd = std(x);      % Evaluate standard deviation of x
```

A MATLAB example that shows the estimation of standard deviation is given next.

EXAMPLE 2.3

Generate a 5000-point array of Gaussianly distributed (i.e., normal) numbers and calculate the standard deviation of those numbers. Plot the histogram of that data set along with a vertical line indicating plus and minus one standard deviation. Repeat with uniformly distributed data, but remove the mean before calculating the standard deviation and plotting.

Solution: Use `randn` to generate the 5000-point array and `std` to calculate the standard deviation. Use `hist` to construct the histogram (see Example 1.4). Plot the histogram and the standard deviation lines. Use `rand` to generate the uniformly distributed data, subtract the mean, and repeat the standard deviation determination and plotting.

```
% Example 2.3 Program to calculate the standard deviation of a set of Gaussian
%  and uniform random numbers, plot the histogram and standard deviations.
%
```

```
N = 5000;                      % Number of data points
nu_bins = 15;                  % Number of bins for histogram
x = randn([1,N]);              % Generate Gaussian random numbers
[H,bin] = hist(x,nu_bins);     % Generate histogram
xsd = std(x);                  % Calculate standard deviation
subplot(2,1,1);                % Select plot
plot(bin,H,'--k'); hold on;    % and plot
plot([xsd xsd],[0 max(H)],'k'); % Plot straight lines
plot([-xsd -xsd],[0 max(H)],'k');
  .........label and title........
x = rand([1,N]);               % Generate uniform random numbers
x = x-mean(x);                 % Remove mean.
  ........repeat standard deviation and plotting.......
```

Results: The plots generated by this example are shown in Figure 2.4. The standard deviation of the Gaussian random numbers is 1.0, which is expected since this is the distribution created by `randn`. The uniformly distributed number random numbers range between ±0.5 (after the mean is removed) and the standard deviation of these numbers is approximately 0.3.

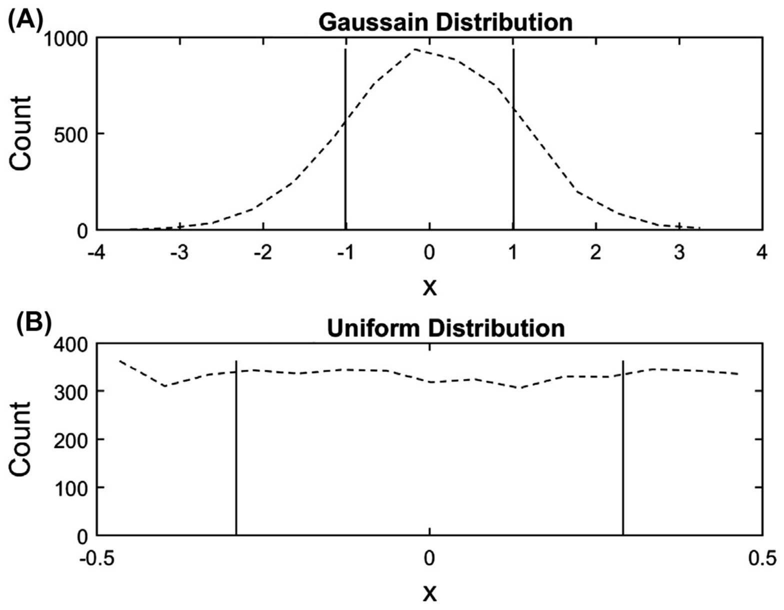


FIGURE 2.4 (A) Histogram of Gaussian random members with vertical lines indicating plus and minus one standard deviation. (B) Histogram of random numbers uniformly distributed between ± 0.5 with standard deviations.

2.2.3 Averaging for Noise Reduction

When multiple measurements are made, multiple values or signals are obtained. If these measurements are combined or added together, the means add so that the combined value or signal has a mean that is the average of the individual means. The same is true for the variance: the variances add and the average variance of the combined measurement is the mean of the individual variances.

$$\overline{s^2} = \frac{1}{N}\sum_{n=1}^{N} s_n^2 \tag{2.16}$$

When a number of signals are added or averaged together, it can be shown that the standard deviations of the noise components are reduced by a factor equal to $1/N$, where N is the number of measurements that are averaged.

$$\overline{s_{Noise}} = \frac{s}{\sqrt{N}} \tag{2.17}$$

In other words, averaging multiple measurements from the same source, will reduce the standard deviation of the measurement's variability or noise by the square root of the number of averages. For this reason, it is common to make multiple measurements whenever possible and average the results. The ability of averaging to reduce noise is demonstrated in the next example of a design simulation.

EXAMPLE 2.4

Use of averaging to reduce noise or measurement variability. You are working for a company that produces a device that measures body temperature using infrared reflection. The basic measurement is known to produce Gaussianly distributed errors with an expected standard deviation of 2.0 degree. In other words, the measurement has Gaussian added noise with a standard deviation of 2.0. You want to demonstrate, using a simulation approach, that by averaging measurements you can effectively reduce the measurement error.

Solution: To accomplish this demonstration, use MATLAB to simulate the measurement with added Gaussian noise having a standard deviation of 2.0 degree. Since the `randn` routine produces a Gaussian distribution with a standard deviation of 1.0, simulate the measurement using:

```
actual_temperature = 98.6;                    % Actual body temperature
x_measured = actual_temperature + 2*randn;    % Simulated measurement
```

To simulate averaging you could use a `for` loop to generate multiple measurements and average these together. Assume you would like to assess averages of 4, 8, 16, and 32.

```
actual_temperature = 98.6  % Actual temperature
N_avg = [4 8 16 32];       % Number of averages
for k1 = 1:4               % Repeat for each average
```

```
  for k = 1:N_avg(k1)
    x_measured(k) = actual_temperature + 2*randn; % Individual observations
   end
  x_avg = mean(x);       % Measurement average
  disp(['Nu_avg: ',num2str(N_avg(k1)),' Avg: ',num2str(x_avg)])  % Display results
end
```

A typical run might output:

```
Nu_avg: 4 Avg: 98.3401
Nu_avg: 8 Avg: 98.575
Nu_avg: 16 Avg: 98.4914
Nu_avg: 32 Avg: 98.6797
```

Since the measurement is random, the averages are random, although we see that the result of 32 averages is closer to the actual temperature than the average of only four individual measurements. However, you want to assess the noise associated with each measurement, that is, the standard deviation of each average. Since this is a simulation, you can just repeat the average measurements a large number of times and calculate the standard deviation of all those averages. This means adding an internal loop leading to the solution:

```
actual_temperature = 98.6;       % Actual value of measurement.
N_avg = [4,8,16,32];             % Number of averages
for k1 = 1:4
  for k2 = 1:100
    for k = 1:N_avg(k1)
      x_measurement(k) = temperature + 2*randn;     % Individual observations
    end
    x_avg(k2) = mean(x_measurement);                % Measurement average
  end
  sd = std(x_avg);         % Average measurement standard deviation
  disp(['Nu_avg: ',num2str(N_avg(k1)),' Std: ',num2str(sd)])   % Display result
end
```

Result: This simulation produces:

```
Nu_avg: 4  Std: 1.1685 deg
Nu_avg: 8  Std: 0.74813 deg
Nu_avg: 16  Std: 0.50366 deg
Nu_avg: 32  Std: 0.33155 deg
```

Since the original noise (i.e., measurement error) had a standard deviation of 2.0 degree, if this noise were reduced by the $\sqrt{N}$, then the resultant noise should be: $2/\sqrt{4} = 1.0$, $2/\sqrt{8} = 0.707$, $2/\sqrt{16} = 0.5$, and $2/\sqrt{32} = 0.354$. Although there is some randomness in the result, this simulation clearly demonstrates that averaging reduces the noise in the final measurement and the reduction is very close to that predicted by Equation 2.17. Since you thought of the idea of using averaging to reduce the noise and then demonstrated to management that it actually works, you can expect a promotion in the near future.

2.2.4 Ensemble Averaging

The use of averaging to reduce noise can be applied to entire signals, a technique known as "ensemble averaging." Ensemble averaging is a simple yet powerful signal processing technique for reducing noise, but you need to be able to make multiple observations on essentially the same system. These multiple observations could come from multiple sensors, but in many biomedical applications they come from repeated responses to the same stimulus. This only works if each signal can be taken as an exact repetition (not counting the added noise) of the other signals. Many biological responses change with repeated stimuli, even if the stimuli are identical (i.e., many biosystems are nonstationary), but a few systems produce highly repetitive responses to even a large number of repeating stimuli.

To construct an ensemble average you must have a time reference, some point that represents a corresponding time in all ensemble signals. For example, the onset time of a repeating stimulus could be used as a timing reference. Figure 2.5 shows three discrete signals

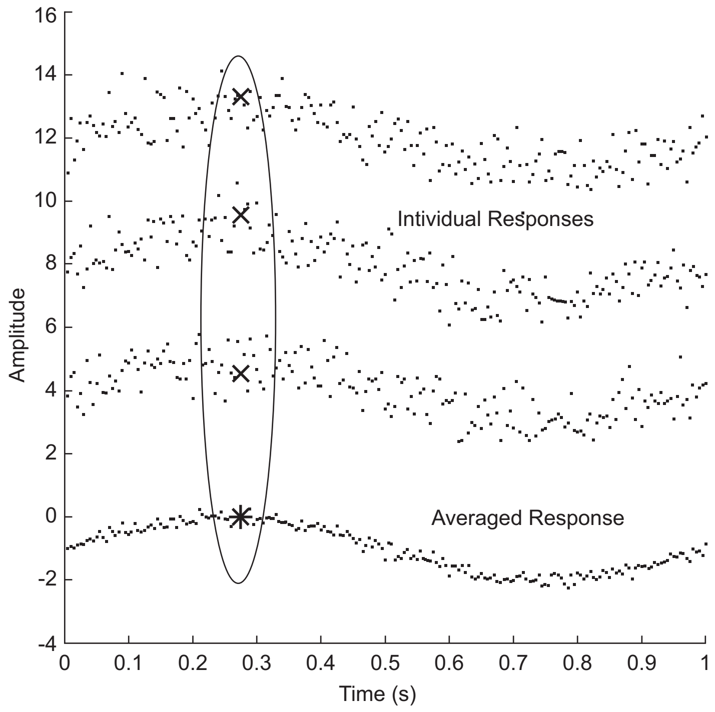


FIGURE 2.5 Three individual signals are used to construct an ensemble average. The signals are shown already aligned to the same time frame. To construct the average response, corresponding points (those that have the same timing with respect to $t = 0$) in the individual responses (x-points) are averaged to make a single point in the averaged response (*-point). This process is repeated for all samples in the individual responses. For these signals, $N = 200$.

consisting of sinusoids and Gaussian noise. (These signals are plotted using separate points to emphasize that they are discrete.) The signals have been aligned so they all share a common time frame. When the individual signals are aligned in time, an ensemble average signal is constructed by averaging over corresponding points from the individual signals. In Figure 2.5, three corresponding points (x-points) are averaged to construct one point in the averaged response (*-point). To construct the entire averaged response, this averaging is repeated for each sample in the time frame. (In the case of Figure 2.5, there are 200 samples in the time frame so the averaging process must be repeated 200 times.) The implementation of ensemble averaging in MATLAB is very easy as illustrated by the example.

A classic biomedical engineering example of the application of ensemble averaging is the visual evoked response (VER) in which a visual stimulus produces a small neural signal embedded in the EEG. Usually this signal cannot be detected in the EEG signal, but by averaging hundreds of EEG signals, time referenced to a visual stimulus, you can determine the visually evoked signal.

To produce an ensemble average, you need multiple response signals and some sort of timing reference that is closely linked to these responses. The timing reference shows how to align the individual responses before averaging. This approach is illustrated in the following example.

EXAMPLE 2.5

Find the average response of a number of individual VERs. The basic signal is an EEG signal recorded near the visual cortex. The stimulus was a repetitive light flash. Individual recordings have too much noise[4] to see the neural response produced by the stimulus. The MATLAB file `ver.mat` contains 100 EEG signals in a matrix variable `ver`. These signals were recorded immediately after the flash and digitized using a sample interval of 0.005 s. (i.e, 5.0 ms). Construct and plot the ensemble average and also plot one of the individual responses to show what the actual EEG signal looked like.

Solution: Use the MATLAB averaging routine `mean`. If this routine is given a matrix variable, it averages each column. If the various signals are arranged as rows in the matrix, the `mean` routine will produce the ensemble average (assuming that they are properly aligned). To determine the orientation of the data, we check the size of the data matrix. Normally the number of signal data points is greater than the number of signals.[5] Since we want the signals to be in rows, if the number of rows is greater than the number of columns, we transpose the data using the MATLAB transposition operator (i.e., the ' symbol).

```
% Example 2.5  Example of ensemble averaging
load ver;                          % Get visual evoked response data;
fs = 1/.005;                       % Sample frequency from sample interval
[nu,N] = size(ver);                % Get data matrix size
if nu > N
       ver = ver';                 % Transpose if necessary
end
t = (1:length(ver))/fs;            % Generate time vector
%
```

```
subplot(1,2,1);
plot(t,ver(3,:));              % Plot  individual  record  (select  record  3
arbitrarily)
      .......label axes.........
% Construct and plot the ensemble average
avg = mean(ver);               % Calculate ensemble average
subplot(1,2,2)
plot(t,avg);                   % Plot ensemble average
.......label axes.........
```

Results: Calculation of the ensemble average could not be easier: only one MATLAB command. But in this example, the hard work, aligning the signals, had already been done. In a practical situation, you might trigger the data acquisition process using the same signal that triggers the light stimulus.

This program produces the results shown in Figure 2.6. Figure 2.6A shows a single response, and the evoked response is not even remotely discernable. In Figure 2.6B the VER from the ensemble average of 100 individual responses is clearly visible.

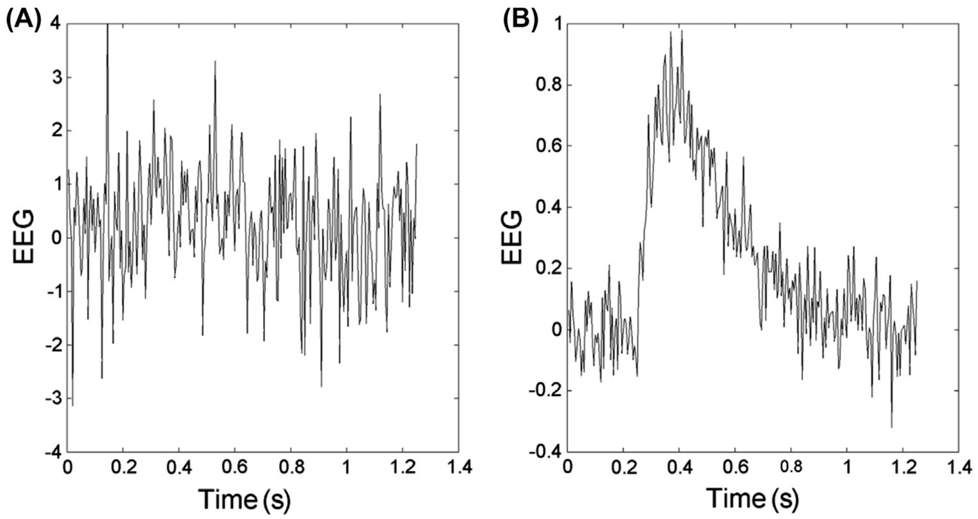


FIGURE 2.6 (A) The raw EEG signal showing a single response to an evoked stimulus. (B) The ensemble average of 100 individual signals showing the VER.

[4]Actually this noise is not really noise but other brain activity, but if it is not of interest, it is noise.
[5]This trick is not fool-proof, but if you guess wrong on the orientation of the signal, it will be obvious in the graphs. You can then try the other orientation.

You might ask if ensemble averaging reduces noise as much as single variable averaging illustrated in Example 2.4. In other words, does the reduction in standard deviation of the averaged signal follow the prediction of Equation 2.17: a reduction proportional to the square root of the number of signals averaged? If so, we would expect that the noise in Figure 2.6B is 10 times less than the raw EEG data in Figure 2.6A. A problem at the end of this chapter explores this question.

2.3 THE BASIC WAVEFORM: SINUSOIDS

In the book *The Martian*, Mark Watney tells us that duct tape does indeed work in a vacuum and that "duct tape is like magic and should be worshiped." As we discover in the next chapter, the unprepossessing sinusoid approaches the same mythic importance in signal analysis. Now we take a closer look at some of the definitions and mathematical operations associated with this important waveform.

2.3.1 Sinusoids as Real-Valued Signals

A sinusoidal signal is any signal that follows a basic sine wave pattern at a single frequency. It can be either real valued or complex (i.e., using complex number representation). Complex sinusoidal signals are described in the next section. Of course all real-world signals are real valued. Describing a sinusoidal signal in the complex domain is mathematically a little more complicated, but surprisingly, complex representation makes sinusoidal math easier.

The most basic representation of a sinusoidal signal was given in Eq. 1.14 in Chapter 1 and is repeated here:

$$x(t) = A\sin(\omega_p t) = A\sin(2\pi f_p t) = A\sin\left(\frac{2\pi t}{T}\right) \tag{2.18}$$

where A is the signal amplitude. Three different methods for describing the sine wave's frequency are shown: ω_p is the frequency in radians per second; f_p is the frequency in hertz; and T is the period in seconds. Frequency in hertz is most common in engineering, but we should be comfortable with any of the representations in Equation 2.18.

To move to a more general sinusoidal representation, we note that sine wave–like signals can also be represented by cosines, and the two are related.

$$\begin{aligned} A\cos(\omega t) &= A\sin\left(\omega t + \frac{\pi}{2}\right) = A\sin(\omega t + 90^\circ) \\ A\sin(\omega t) &= A\cos\left(\omega t - \frac{\pi}{2}\right) = A\cos(\omega t - 90^\circ) \end{aligned} \tag{2.19}$$

The right-hand representations (i.e., $A\sin(\omega t + 90^\circ)$ and $A\cos(\omega t - 90^\circ)$) have confused units. The first part of the sine argument, ωt, is in radians, whereas the second part is in degrees. Nonetheless, this is fairly common usage and we use it a lot; however, in MATLAB the trig function arguments must be in radians. To convert between degrees and radians. note that 2π radians = 360 degrees, so 1 radian = $360/2\pi$ degrees, and 1 degree = $2\pi/360$ radians.

A general real-valued sinusoidal signal is just a sine or cosine with a nonzero phase term:

$$
\begin{aligned}
x(t) &= A\sin(2\pi ft+\theta) = A\sin(\omega t+\theta) = A\sin\left(\frac{2\pi t}{T}+\theta\right) \quad \text{or} \\
x(t) &= A\cos(2\pi ft+\theta) = A\cos(\omega t+\theta) = A\cos\left(\frac{2\pi t}{T}+\theta\right)
\end{aligned}
\tag{2.20}
$$

where again the phase θ is usually expressed in degrees, even though the frequency descriptor (ω, or $2\pi ft$, or less common $2\pi t/T$) is expressed in hertz or radians. Many of the sinusoidal signals we use are versions of Equation 2.20. Note that to completely describe a sinusoid, we need only three numbers: amplitude A, phase θ, and frequency f (or ω). Example 2.6 uses MATLAB to generate two sinusoids that differ by 60 degree.

EXAMPLE 2.6

Use MATLAB to generate and plot two sinusoids that are 60 degrees out of phase. Calculate, analytically, the difference in time between the two sinusoids assuming they have a frequency of 2.0 Hz.

Solution, MATLAB: Since no details are given except the sinusoidal frequency, this example is mainly an exercise in specifying signal parameters. For sampling frequency and data length we arbitrarily select an f_s of 1.0 kHz and N of 500. The 500 data points are certainly enough to generate a smooth plot and with an f_s of 1.0 kHz we get a signal length of 0.5 s. This will produce 1 cycle of our 2 Hz sinusoids, which should make for a nice plot.

We can generate one sinusoid using the techniques in Examples 1.6 and 2.1 with zero phase angle (i.e., $\theta = 0$ in Equation 2.20). The second sinusoid is similarly generated, except that we make the phase 60 degree. (i.e., $\theta = 60$ in Equation 2.20). We do need to convert the phase from deg to rad for MATLAB. Since 1 degree = $2\pi/360$ radians, we simply let MATLAB multiply our 60 degree by `2*pi/360`.

```
% Example 2.6 Plot two sinusoids 60 deg out of phase
%
fs = 1000;                 % Assumed sampling frequency
N = 500;                   % Number of points
t = (0:N-1)/fs;            % Generate time vector
f = 2;                     % Sinusoid frequency
phase = 60*(2*pi/360);     % 60 deg phase; convert to radians
x1 = sin(2*pi*f*t);        % Construct sinusoids
x2 = sin(2*pi*f*t + phase); % One with a 60 phase shift
hold on;
plot(t,x1);                % Plot the unshifted sinusoid
plot(t,x2,'--');           % Plot using a dashed line
plot([0 .5], [0 0]);       % Plot horizontal line
  .......label axes.......
```

Results, MATLAB: The graph produced by this simple example program is shown in Figure 2.7.

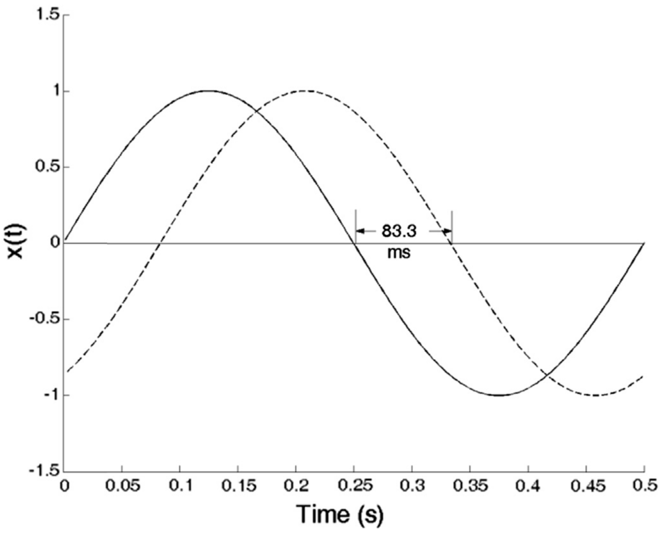


FIGURE 2.7 Two sinusoids with a phase difference of 60 degree. For 2 Hz sinusoids that translates to a time difference of 83 ms.

Solution, Analytical: To convert the difference in phase angle to a difference in time, note that the phase angle varies through 360 degree during the course of one period, and a period is T sec long. Hence to calculate the time difference between the two sinusoids given the phase angle θ:

$$t_d = \frac{\theta}{360}T = \frac{\theta}{360f} \tag{2.21}$$

For 2 Hz sinusoids, $T = 0.5$ s, so substituting T into Equation 2.21:

$$t_d = \frac{\theta}{360}T = \frac{60}{360}0.5 = 0.0833 \text{ sec.} = 83.3\, m \text{ sec.}$$

EXAMPLE 2.7

Find the time difference between two sinusoids:

$$x_1(t) = \cos(4t + 30), \text{ and } x_2(t) = 2\sin(4t)$$

Solution: This has the added complication that one waveform is defined in terms of the cosine function and the other in terms of the sine function. So we need to convert them both to either a sine or cosine. Here we convert to cosines:

$$x_2(t) = 2\cos(4t - 90)$$

So the total angle between the two sinusoids is $30 - (-90) = 120$ degree and since the period is:

$$T = \frac{1}{f} = \frac{2\pi}{\omega} = \frac{2\pi}{4} = 1.57 \text{ sec.}$$

and the time delay is:

$$t_d = \frac{\theta}{360}T = \frac{120}{360}1.57 = 0.523 \text{ sec.}$$

Equation 2.20 describes a fairly intuitive way of thinking about a sinusoid as a sine wave with a phase shift. Alternatively, a cosine could just as well be used instead of the sine as also shown in Equation 2.20. Here we will use both.

Sometimes it is mathematically convenient to represent a sinusoid as a combination of a pure sine and a pure cosine rather than a single sinusoid with a phase angle. This representation can be achieved using the well-known trigonometric identity for the difference of two arguments of a cosine function:

$$\cos(x - y) = \cos(x)\cos(y) + \sin(x)\sin(y) \tag{2.22}$$

Based on this identity, the equation for a sinusoid can be written as:

$$\begin{aligned} C\cos(2\pi ft - \theta) &= C\cos(\theta)\cos(2\pi ft) + C\sin(\theta)\sin(2\pi ft) \\ &= a\cos(2\pi ft) + b\sin(2\pi ft) \end{aligned} \tag{2.23}$$

where:

$$a = C\cos(\theta) \text{ and } b = C\sin(\theta) \tag{2.24}$$

Note that θ is defined as negative in these equations. To convert the other way, from a sine and cosine to a single sinusoid with amplitude C and angle θ, start with Equation 2.24.

$a = C\cos(\theta)$ and $b = C\sin(\theta)$: to determine C, square both equations and sum:

$$\begin{aligned} a^2 + b^2 &= C^2\left(\cos^2\theta + \sin^2\theta\right) = C^2 \\ C &= \sqrt{a^2 + b^2} \end{aligned} \tag{2.25}$$

The calculation for θ given a and b is:

$$\begin{aligned} \frac{b}{a} &= \frac{C\sin(\theta)}{C\cos(\theta)} = \tan(\theta) \quad \text{and...} \\ \theta &= \tan^{-1}\left(\frac{b}{a}\right) \end{aligned} \tag{2.26}$$

Again, Equation 2.26 finds $-\theta$ as defined in Equation 2.23.

Care must be taken in evaluating Equation 2.26 to ensure that θ is determined to be in the correct quadrant based on the signs of a and b. Many calculators and MATLAB's `atan` function will evaluate θ as in the first quadrant (0−90 degree) if the ratio of b/a is positive, and in the fourth quadrant (270−360 degree) if the ratio is negative. If both a and b are positive, then

θ is between 0 and 90 degree, but if b is positive and a is negative, then θ is in the second quadrant between 90 and 180 degree, and if both a and b are negative, θ must be in the third quadrant between 180 and 270 degree. Finally, if b is negative and a is positive, then θ must belong in the fourth quadrant between 270 and 360 degree. You can avoid all this worry by using MATLAB's `atan2(b,a)` routine, which correctly outputs the arctangent of the ratio of `b` over `a` (in radians) in the appropriate quadrant.

To add sine waves simply add their amplitudes. The same applies to cosine waves:

$$\begin{aligned} a_1\cos(\omega t) + a_2\cos(\omega t) &= (a_1 + a_2)\cos(\omega t) \\ a_1\sin(\omega t) + a_2\sin(\omega t) &= (a_1 + a_2)\sin(\omega t) \end{aligned} \tag{2.27}$$

To add two sinusoids (i.e., $C\sin(\omega t + \theta)$ or $C\cos(\omega t - \theta)$), convert them to sines and cosines using Equation 2.23, add sines to sines and cosines to cosines (Equation 2.27), and convert back to a single sinusoid if desired.

EXAMPLE 2.8

Convert the sum of a sine and cosine wave, $x(t) = -5\cos(10t) - 3\sin(10t)$ into a single sinusoid.

Solution: Apply Equations 2.25 and 2.26

$$a = -5 \quad \text{and} \quad b = -3$$

$$C = \sqrt{a^2 + b^2} = \sqrt{(-5)^2 + (-3)^2} = 5.83$$

$$\theta = \tan^{-1}\left(\frac{b}{a}\right) = \tan^{-1}\left(\frac{-3}{-5}\right) = 31 \ \text{deg},$$

But θ must be in the third quadrant since both a and b are negative:

$$\theta = 31 + 180 = 211 \ \text{deg}.$$

Therefore, the single sinusoid representation would be:

$$x(t) = C\cos(\omega t - \theta) = 5.83\cos(10t - 211^\circ)$$

Using the above-mentioned equations, any number of sines, cosines, or sinusoids can be combined into a single sinusoid provided they are all at the same frequency. This is demonstrated in Example 2.9.

EXAMPLE 2.9

Combine $x(t) = 4\cos(2t + 30^\circ) + 3\sin(2t + 60^\circ)$ into a single sinusoid.

Solution: First expand each sinusoid into a sum of cosine and sine, then algebraically add the cosines and sines, then recombine them into a single sinusoid. Be sure to convert the sine into a cosine (recall Equation 2.19: $\sin(\omega t) = \cos(\omega t - 90^\circ)$) before expanding this term.

$$\begin{aligned} &4\cos(2t + 30) = a\cos(2t) + b\sin(2t) \\ &\text{where:} \quad a = C\cos(\theta) = 4\cos(-30) = 3.5 \quad \text{and} \quad b = C\sin(\theta) = 4\sin(-30) = -2 \\ &4\cos(2t + 30) = 3.5\cos(2t) - 2\sin(2t) \end{aligned}$$

(Note that θ is actually negative in the above equations.)

Convert the sine to a cosine then decompose the sine into a cosine plus a sine:

$$3\sin(2t + 60) = 3\cos(2t - 30) = 2.6\cos(2t) + 1.5\sin(2t)$$

Combine cosine and sine terms algebraically:

$$4\cos(2t + 30) + 3\sin(2t + 60) = (3.5 + 2.6)\cos(2t) + (-2 + 1.5)\sin(2t)$$

$$= 6.1\cos(2t) - 0.5\sin(2t)$$

$$= C\cos(2t + \theta) \quad \text{where}: \; C = \sqrt{6.1^2 + (-0.5)^2} \quad \text{and} \quad \theta = \tan^{-1}\left(\frac{-.5}{6.1}\right)$$

$$C = 6.1; \quad \theta = 4.7 \; (\text{Since } b \text{ is negative, } \theta \text{ is in 4th quadrant}) \text{ so } \theta = -4.7 \text{ deg}$$

$$x(t) = 6.1\cos(2t - 4.7^\circ)$$

This approach can be extended to any number of sinusoids as shown in the problems.

2.3.2 Complex Representation of Sinusoids

Why use complex numbers? If you are willing to deal with complex numbers and variables (and you are), sinusoidal representation becomes a lot simpler. The beauty of complex numbers and complex variables is that the real and imaginary parts are orthogonal. Orthogonality is discussed later, but the importance for complex numbers is that the real and imaginary parts do not interact; they are independent as we explain in the next paragraph.

Recall that a complex number combines a real number and an imaginary number. Real numbers are what we commonly use, whereas imaginary numbers are the result of square roots of negative numbers and are represented as real numbers multiplied by $\sqrt{-1}$. In mathematics, $\sqrt{-1}$ is represented by the letter i, but engineers use the letter j, reserving the letter i for current. So, although 5 is a real number, $j5$ (i.e., $5\sqrt{-1}$) is an imaginary number. A complex number is just a combination of a real and imaginary number such as $5 + j5$. A complex variable simply combines a real and an imaginary variable: $z = x + jy$. The arithmetic of complex numbers and some of their important properties are reviewed in Appendix E. We use complex variables and complex arithmetic extensively in later chapters, so it is worthwhile to review these operations.

Complex numbers are illustrated graphically as inhabiting orthogonal horizontal and vertical axes as shown in Figure 2.8. Clearly you can change one component of a complex number without changing the other. This means that a complex number is really two separate numbers rolled into one and a complex variable is two separate variables.

Getting two numbers for the price of one would not be such a big deal except that for a given frequency, f, we only need two numbers to completely describe a sinusoid: A and θ in Equation 2.20 or a and b in Equation 2.23. This suggests that it might be possible to describe a sinusoid with only a single complex number. Thanks to the Swiss mathematician

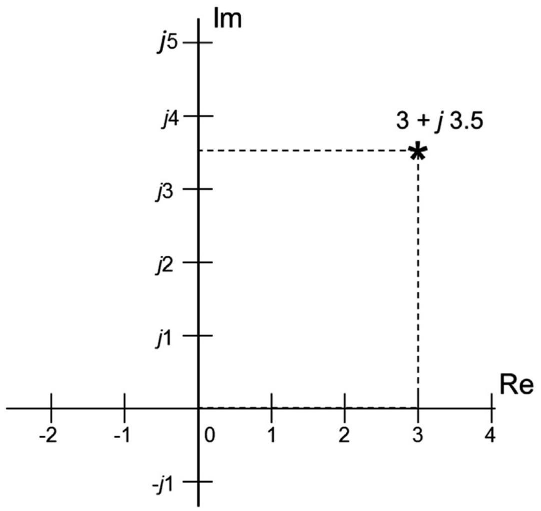


FIGURE 2.8 A graphical representation of real and imaginary numbers that illustrates their orthogonality, or independence. A change in the value of an imaginary number does not affect the real number and vice versa.

Leonhard Euler (pronounced "oiler"),[6] we know how do to just that. Euler's identity relates a complex exponential to a real cosine and imaginary sine.[7]

$$e^{\pm jx} = \cos x \pm j \sin x \tag{2.28}$$

Note that this equation looks like the representation for a sinusoid given by Equation 2.23 except that the sine term is imaginary. Although this difference necessitates some extra mathematical features, it is well worth the result: the ability to represent a sinusoid by a single number or variable.

MATLAB variables can be either real or complex and MATLAB deals with them appropriately. A typical MATLAB number is identified by its real and imaginary part: `x = 2 + 3i` or `x = 2 + 3j`. MATLAB assumes that both `i` and `j` stand for the complex operator unless they are defined to mean something else in your program. Another way to define a complex number is using the `complex` routine: `z = complex(x,y)` where `x` and `y` are two arrays. The first argument, `x`, is taken as the real part of `z` and the second, `y`, is taken as the imaginary part of `z`. To find the real and imaginary components of a complex number or array, use `real(z)` to get the real component and `imag(z)` to get the imaginary component. To find the polar components of a complex variable use `abs(z)` to get the magnitude component and `angle(z)` to get the angle component (in radians).

[6]The use of the symbol *e* for the base of the natural logarithmic system is a tribute to Euler's extraordinary mathematical contributions.

[7]The derivation for this equation is given in Appendix A.

EXAMPLE 2.10

Demonstration of the Euler equation in MATLAB. Generate one cycle of the complex sinusoid given in Equation 2.28 using a 1.0-s time vector. Assume $f_s = 500$ Hz. Plot the real and imaginary parts of this function superimposed to produce a cosine and sine wave.

Solution: Construct the time vector given the sample frequency and total desired time. Although you are not given the number of points the time vector is easily determined by constructing a vector between 0 and 1.0 with steps of $1/f_S$: `t = (0:1/fs:1.0);` . Then generate e^{-jx} as in Equation. 2.28 where $x = 2\pi t$ so that the complex sinusoid is one cycle (x will range from 0 to 2π as t goes from 0 to 1.0). Plot the real and imaginary part using MATLAB's `real` and `imag` routines.

```
% Example 2.10 Demonstration of the Euler equation in MATLAB
%
fs = 500;                             % Sampling frequency
t = (0:1/fs:1);                       % Time vector
z = exp(-j*2*pi*t);                   % Complex sinusoid (Equation 2.28)
plot(t,real(z),'k',t,imag(z),':k');   % Plot result
  .......labels........
```

Result: The sine and cosine wave produced by this program are seen in Figure 2.9.

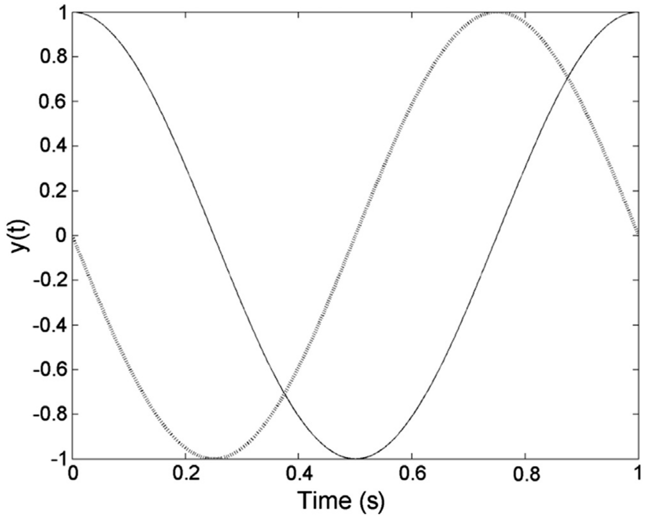


FIGURE 2.9 A sine wave and cosine wave produced from a single complex variable using Euler's identity (Equation 2.28).

The convenience of representing a sinusoid by a single complex exponential will find important applications in two sinusoidally based analysis techniques: Fourier analysis described in the next chapter and phasor analysis described in Chapter 6.

2.4 TIME DOMAIN ANALYSIS

The basic measurements of mean, variance, standard deviation, and rms generally do not capture the important features of a signal. For example, if we have digital versions of the EEG signals shown in Figure 2.2, we can easily compute their mean, variance, standard deviation, and rms value, but these would not tell us much about the signals or the neural processes that created them. More insight might be gained by comparing these signals with some reference signal(s).

Comparing a signal with a reference signal, or perhaps a group or "family" of reference signals, is an oft used tool in signal analysis. Such reference signals, or signal families, tend to be much less complicated than the signal such as a sine wave or family of sine waves.[8] A quantitative comparison can tell you how much your complicated signal is like a simpler, easier to understand reference signal or family. If enough comparisons are made with a well-chosen reference family, these comparisons can actually provide an alternative representation of the signal. Sometimes this new representation of the signal is more informative or enlightening than the original.

2.4.1 Comparison Through Correlation

We are constantly making subjective comparisons in shopping, job hunting, dating, and many other aspects of daily life, but in signal analysis we need objective, quantitative comparisons. We need to get a number that describes how well one signal compares with another. One of the most common ways of quantitatively comparing two functions is through mathematical correlation. Applied to signals, correlation seeks to quantify how much one signal is like another. This sounds like just what we need, and mathematical correlation does a pretty good job of describing similarity, but once in a while it breaks down. For example, a sine wave and cosine wave are pretty much alike in that they both exhibit similar oscillatory behavior, yet in Figure 2.10 we show their mathematical correlation is zero.

Correlation between different signals is illustrated in Figure 2.10, which shows various pairs of waveforms and the correlation between them. The lack of correlation between two sinusoids shows that correlation does not always measure general similarity. Mathematically they are as un-alike as possible, even though they have similar behavioral patterns.

The linear correlation between two digital functions or signals can be obtained using the Pearson correlation coefficient defined as:

$$r_{xy} = \frac{1}{(N-1)s_x s_y} \sum_{n=1}^{N} (x_n - \overline{x})(y_n - \overline{y}) \tag{2.29}$$

where r_{xy} is a common symbol in signal analysis for the correlation between x and y. (For the Pearson correlation coefficient, the symbol ρ_{xy} is also used.) The variables x and y could represent any two waveforms. Again $\overline{x}$ and $\overline{y}$ are the means of x and y (Equations 2.3 and 2.4),

[8]Sine waves are pretty simple. We know they can be completely described by only three numbers. In the next chapter, we show that sine waves are simple in other respects; they contain energy at only one frequency.

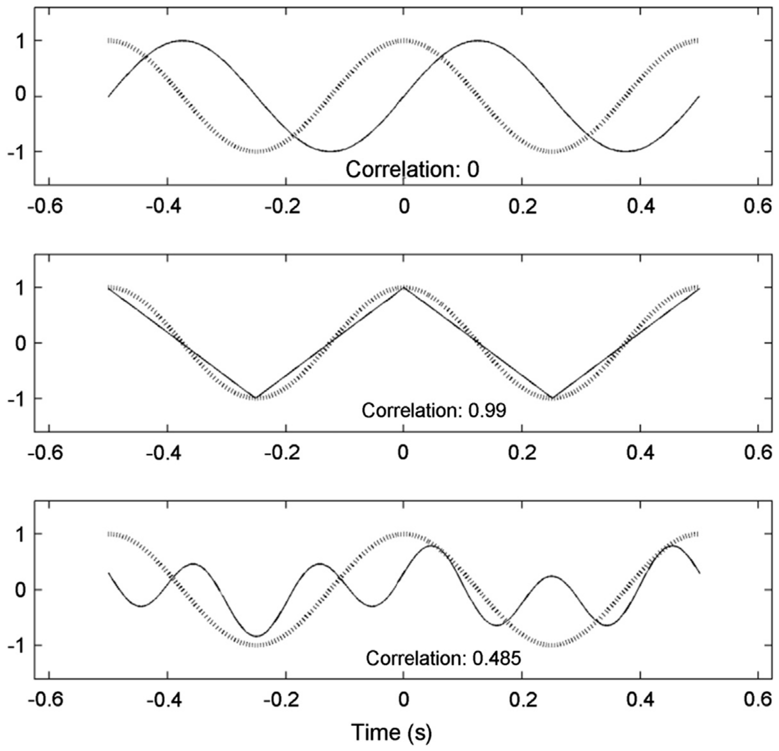


FIGURE 2.10 Three pairs of signals and the correlation between them as given by the Pearson correlation coefficient defined in Equation 2.29. The high correlation between the sine wave and triangular wave (center) correctly expresses the similarity between them, but the zero correlation between the sinusoids in the upper plot does not reflect their general similarity.

and s_x and s_y are the standard deviations of x and y (Equations 2.14 and 2.15). This equation scales the correlation coefficient to be between ±1, but if we are not concerned with the scale, then it is not necessary to normalize by the standard deviations and $N - 1$. If, in addition, the means of the signal and reference are zero, correlations can be determined using a simpler equation:

$$r_{xy} = \frac{1}{N}\sum_{n=1}^{N} x[n]y[n]$$

$$r_{xy} = \frac{1}{N}\Big(x[1]y[1] + x[2]y[2] + x[3]y[3]\ldots + x[N]y[N]\Big) \tag{2.30}$$

where r_{xy} is the unscaled correlation between x and y. Since this is a relative correlation, the summation can be used directly without scaling by $1/N$. This is the basic normalized correlation. If continuous functions are involved, the summation becomes an integral and the discrete functions, $x[n]$ and $y[n]$, become continuous functions, $x(t)$ and $y(t)$:

$$r_{xy} = \frac{1}{T}\int_0^T x(t)y(t)dt \tag{2.31}$$

Both Equations 2.30 and 2.31 use integration (or summation) and scaling (dividing by T or N) to get the average of the product between signals over some range. The correlation values produced by Equations 2.30 and 2.31 will not range between ± 1, but they do give relative values that are proportional to the linear correlation of x and y. The correlation of r_{xy} will have the largest possible positive value when the two functions are identical and the largest negative value when the two functions are exact opposites of one another (i.e., one function is the negative of the other). The average product produced in these equations is zero when the two functions are mathematically completely dissimilar.

The only difference between the Pearson correlation coefficient given in Equation 2.29 and the correlation given in Equation 2.30 is the normalization that makes the correlation values range between ± 1.0. To convert an unnormalized correlation to a Pearson normalization:

$$r_{xy\ Pearson} = \frac{r_{xy\ un-norm}}{(N-1)s_x s_y} = \frac{r_{xy-un-norm}}{(N-1)\sqrt{{s_x}^2 {s_y}^2}} \tag{2.32}$$

where the standard deviations, s, are defined in Equations 2.14 and 2.15. Note that we can also use the square root of the variances s^2, as defined in Equations 2.12 and 2.13. The term "correlation coefficient" implies this Pearson normalization, whereas the term "correlation" is used more loosely and could mean normalized or unnormalized correlation.

Some deeper concepts are associated with correlation and vectors. Recall that in Chapter 1, it is mentioned that a signal sequence made up of n numbers could be thought of as a vector in n-dimensional space. Running with this concept, correlation becomes the projection of one signal vector upon the other: it is the mathematical equation for projecting an n-dimension x vector upon an n-dimension y vector. When two functions are uncorrelated, they are also said to be orthogonal and their n-dimensional vectors are perpendicular. Such vectors have a projection of zero on one another. In fact, a good way to test if two functions are orthogonal is to evaluate their correlation. When this concept of vector projection is used to describe correlation, it is common to refer to a family of reference signals or functions as a *basis* or *basis functions*. Hence "projecting a signal on a basis" is just a cool way of saying "correlating a signal with a reference family." Moreover, even if you like thinking of signals as n-dimensional vectors, you still use Equations 2.29 or 2.30 to compute their correlation.

Covariance computes the variance that is shared between two (or more) signals. Covariance is usually defined in discrete notation as:

$$Cov = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \overline{x})(y_n - \overline{y}) \tag{2.33}$$

The equation for covariance is similar to the discrete form of the Pearson correlation except that it is not normalized by $\frac{1}{s_x s_y}$.

As seen from the above-mentioned equations, correlation operations have both continuous and discrete versions so that both analytical and computer solutions are possible. The next two examples solve for the correlation between two waveforms using both the continuous, analytical approach and a computer algorithm.

EXAMPLE 2.11

Use Equation 2.31 (continuous form) to find the correlation (unnormalized) between the sine wave and the square wave shown in Figure 2.11A. Also find the correlation between a cosine wave and a square wave shown in Figure 2.11B. All waveforms have amplitudes of 1.0 V (peak-to-peak) and periods of 4.0 s.

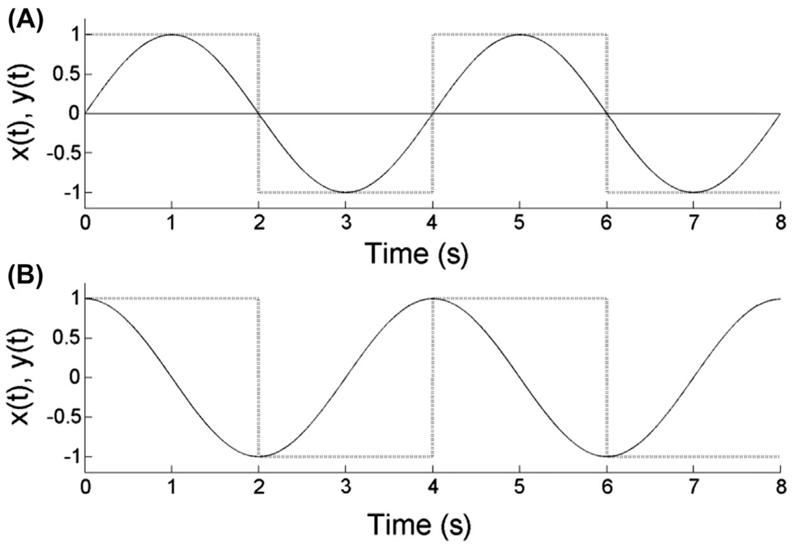


FIGURE 2.11 (A) Sine wave and square wave. (B) Cosine wave and square wave. The unbiased correlation between these waveforms is found in Example 2.11.

Solution. Sine and square wave. Apply Equation. 2.31, but use symmetry to make it easier. Since the waveforms are periodic, we only need evaluate over one period. Moreover, the correlation in the second half of the 1-s period equals the correlation in the first half, so it is only necessary to calculate the correlation period in the first half.

$$Corr = \frac{1}{T}\int_0^T x(t)y(t)dt = \frac{2}{T}\int_0^{T/2}(1)\sin\left(\frac{2\pi t}{T}\right)dt = \frac{2}{T}\frac{T}{2\pi}\left(-\cos\left(\frac{2\pi t}{T}\right)\right)\Bigg|_0^{T/2}$$

$$Corr = \frac{1}{\pi}(-\cos(\pi) - -\cos(0)) = \frac{2}{\pi}$$

Solution. Cosine and square wave. A good look at two waveforms shows that the correlation in the first half is equal and opposite to that in the second half, so we can guess the total correlation is zero. But to have more fun with basic calculus, we work it all out, calculating the correlation over both half cycles.

$$Corr = \frac{1}{T}\int_0^T x(t)y(t)dt = \frac{1}{4}\left(\int_0^2 (1)\cos\left(\frac{2\pi t}{4}\right)dt + \int_2^4 (-1)\cos\left(\frac{2\pi t}{4}\right)dt\right)$$

$$Corr = \frac{1}{4}\left(\frac{4}{2\pi}\left(\sin\left(\frac{2\pi t}{4}\right)\right)\Bigg|_0^2 - \frac{4}{2\pi}\left(\sin\left(\frac{2\pi t}{4}\right)\right)\Bigg|_2^4\right)$$

$$Corr = \frac{1}{2\pi}(\sin(\pi) - \sin(0) - \sin(2\pi) + \sin(\pi)) = \frac{1}{2\pi}(0 - 0 - 0 + 0) = 0$$

Analysis: We find correlation between a sine and square wave but none between a cosine and square wave. This is another example of how correlation does not always represent similarity. We find out how to get around this problem using a technique called "cross-correlation" (see Section 2.4.4).

The next example uses MATLAB to confirm the analytical results found in Example 2.11.

EXAMPLE 2.12

Use Equation 2.31 and MATLAB to confirm the result given in Example 2.11.

Solution: Construct a time vector 500 points long ranging from 0.0 to 4 s (construct only one period of the waveforms.). Use the time vector to construct a 0.25 Hz sine wave and a cosine wave. Then evaluate the correlation between the two by direct application of Equation 2.31.

```
% Example 2.12 Confirm the results of Example 2.11.
N = 500;                           % Number of points
Tt = 4.0                           % Desired total time
f = 0.25;                          % Wave frequency in Hz
fs = N/Tt;                         % Calculate sampling frequency
t = (0:N-1)/fs;                    % Time vector from 0 (approx.) to 4 sec
x = sin(2*pi*f*t);                 % 0.25 Hz sine wave
y = cos(2*pi*f*t);                 % 0.25 Hz cosine wave
z = [ones(1,N/2) -ones(1,N/2)];    % 0.25 Hz square wave
rxz = mean(x.*z);                  % Correlation (Equation 2.30) x and z
rxy = mean(x.*y);                  % Correlation x and y
disp([rxz rxy])                    % Output correlations
```

Analysis: In Example 1.3 we learned how to construct a time vector given the number of points and sampling frequency (N and f_s) or the total time and sampling frequency (T_t and f_s). Here we are presented with a third possibility, the number of points and total time (N and T_t). To calculate this time vector, the appropriate sampling frequency was determined from the total points and total time: $f_s = N/T_t$. Note that Equation 2.30 takes a single line of MATLAB code.

Result: The MATLAB program produces:

```
rxz = 0.63659    rxy = -3.1627e-017
```

The correlation between the sine and square wave, `rxz`, is very close to that found analytically: $2/\pi = 0.6366$. The correlation of the square and cosine wave, `rxy` is close to zero. This is an example where a hand calculation is more accurate than the computer since we know the true correlation should be 0.0. Computer round-off errors result in a small, meaningless, negative correlation.

2.4.2 Orthogonal Signals and Orthogonality

Orthogonal signals and functions can be very useful signal processing tools. In common usage, "orthogonal" means perpendicular: if two lines are orthogonal, they are perpendicular. In the graphical representation of complex numbers shown in Figure 2.8, the real and imaginary components are perpendicular to one another; hence, they are orthogonal. But what makes two signals orthogonal? The formal definition for orthogonal signals is that their inner product (also called the dot product) is zero:

$$\int_{-\infty}^{\infty} x(t)y(t)dt = 0 \tag{2.34}$$

For discrete signals, this becomes:

$$\sum_{n=1}^{N} x[n]y[n] = 0 \tag{2.35}$$

These equations[9] are the same as the correlation equations (Equations 2.30 and 2.31); the missing normalization by $1/N$ could be multiplied out since one side is zero. So signals that are orthogonal are uncorrelated and vice versa, and any correlation method, the Pearson correlation or the unnormalized versions, could test if two signals are orthogonal.

An important characteristic of signals that are orthogonal (i.e., uncorrelated) is that when they are combined or added together they do not interact with one another. This noninterference comes in handy when we represent a complicated signal with a collection of more basic signals. If the basic signals are orthogonal, you can determine each one independently without worrying about the other signals in the collection. Orthogonality simplifies many calculations where multiple signals are involved. Some analyses could not be done, at least not practically, using nonorthogonal signals. Orthogonality is not limited to just two signals. Whole families exist where each signal is orthogonal to all other members in the family. Such families of orthogonal signals are called "orthogonal sets."

2.4.3 Correlations Between Signals

MATLAB has functions for determining the correlation or covariance that are particularly useful when several signals are compared. A matrix of correlations among multiple signals can be calculated using `corrcoef`. Similarly, for covariances use `cov`. The covariance is the correlation normalized by N given by Equation 2.30, whereas the matrix of correlations is the Pearson correlation normalized as in Equation 2.29. For the latter case, correlation values

[9]In Chapter 1 and in the discussion of correlation, we mentioned that a digital signal composed of N samples can be considered a single vector in N-dimensional space. In this representation, two orthogonal signals would have vectors that actually are perpendicular. So the concept of orthogonal holds for the vector representation of signals as well.

run between ± 1, with 1.0 indicating identical signals and −1.0 indicating identical signals, but with one inverted with respect to the other. The calls are similar for both functions:

```
Rxx = corrcoef(X);        % Signal correlations
S = cov(X);               % Signal covariances
```

where `X` is a matrix that contains the various signals to be compared in columns. Some options are available as explained in the associated MATLAB help file. The output, `Rxx`, of `corrcoef` is an n-by-n matrix where n is the number of signals (i.e., columns). The diagonals of this matrix represent the Pearson correlation of the signals with themselves and therefore must be 1.0. The off-diagonals represent the Pearson correlation coefficients of the various combinations. For example, r_{12} is the correlation between signals 1 and 2. Since the correlation of signal 1 with signal 2 is the same as that of signal 2 with signal 1, $r_{12} = r_{21}$, and in general $r_{\mathrm{m,n}} = r_{\mathrm{n,m}}$, so the matrix will be symmetrical about the diagonals:

$$r_{x,y} = \begin{bmatrix} 1 & r_{1,2} & \cdots & r_{1,N} \\ r_{2,1} & 1 & \cdots & r_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{N,1} & r_{N,2} & \cdots & 1 \end{bmatrix} \tag{2.36}$$

where N is the number of signals being compared.

The `cov` routine produces a similar output, except the diagonals are the variances of the various signals and the off-diagonals are the covariances, the same values given by Equation 2.33.

$$S = \begin{bmatrix} \mathrm{cov}_{1,1} & \mathrm{cov}_{1,2} & \cdots & \mathrm{cov}_{1,N} \\ \mathrm{cov}_{2,1} & \mathrm{cov}_{2,1} & \cdots & \mathrm{cov}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}_{N,1} & \mathrm{cov}_{n,2} & \cdots & \mathrm{cov}_{N,N} \end{bmatrix} \tag{2.37}$$

Example 2.13 uses covariance and correlation analysis to determine if sines and cosines of different frequencies are orthogonal. Again, two orthogonal signals have zero correlation. Either covariance or correlation could be used to determine if signals are orthogonal. Example 2.13 uses both.

EXAMPLE 2.13

Determine if 1.0 Hz sine and cosine waves are orthogonal and if a 2.0 Hz cosine wave is orthogonal to a 1.0 Hz cosine wave. Of course, we already know the answer, but this example shows how well the `cov` and `corrcoef` routines perform. The 1 and 2 Hz cosine waves are called "harmonically related," as they have frequencies that are multiples. Also determine if a 1.0 Hz sawtooth is orthogonal to the sinusoidal waveforms. Make the peak-to-peak amplitude of the four signals ±1.0.

Solution: Generate a 1000-point, 1.0-s time vector using the approach in Example 2.12. Use this time vector to generate a data matrix with four columns representing the 1.0 Hz cosine and sine waves, the 2.0 Hz sine wave, and the 1.0 Hz sawtooth. To generate a sawtooth, use the MATLAB routine `sawtooth(2*pi*t)`, to generate a 1.0 Hz sawtooth wave. Apply the covariance and correlation MATLAB routines (i.e., `cov` and `corrcoef`) and display results.

```
% Example 2.13
% Application of the covariance matrices to sinusoids that
% are orthogonal and a sawtooth
%
N = 1000;                % Number of points
Tt = 1;                  % desired total time
fs = N/Tt;               % Calculate sampling frequency
t = (0:N-1)/fs;          % Time vector from 0 (approx.) to 1 sec
X(:,1) = cos(2*pi*t)';   % Generate a 1 Hz cosine
X(:,2) = sin(2*pi*t)';   % Generate a 1 Hz sine
X(:,3) = cos(4*pi*t)';   % Generate a 2 Hz cosine
X(:,4) = saw(2*pi*t)';   % Generate a 1 Hz sawtooth
%
S = cov(X)               % Print covariance matrix
Rxx = corrcoef(X)        % and correlation matrix
```

Analysis: The program defines a time vector in the standard manner. The program then generates the three sinusoids using this time vector in conjunction with `sin` and `cos` functions, arranging the signals as columns of `X`. The program then uses the `sawtooth` routine to generate a 1.0 Hz sawtooth having the same number of samples as the sinusoids. The four signals are placed in a single matrix variable `X`. The program then determines the covariance and correlation matrices of `X`.

Results: The output from this program is a covariance and correlation matrix. The covariance matrix is:

```
S =
   0.5005  -0.0000  -0.0000  -0.0010
  -0.0000   0.5005   0.0000   0.3186
  -0.0000   0.0000   0.5005   0.0000
  -0.0010   0.3186   0.0000   0.3337
```

and

```
Rxx =
   1.0000  -0.0000  -0.0000  -0.0024
  -0.0000   1.0000   0.0000   0.7797
  -0.0000   0.0000   1.0000  -0.0024
  -0.0024   0.7797  -0.0024   1.0000
```

The diagonals of the covariance matrix give the variance of the four signals. These are consistent for the sinusoids and slightly less for the sawtooth. The correlation matrix shows similar results except that the diagonals are now 1.0 since these reflect the correlation of the signals with themselves.

The covariance and correlation between the various signals are given by the off-diagonals. The off-diagonals show that the sinusoids have no correlation with one another, a reflection of their orthogonal relationship. The 1.0 Hz sine wave is correlated with the sawtooth, but there is negligible correlation between the sawtooth and the cosine wave or the 2.0 Hz cosine wave.

2.4.4 Shifted Correlations: Cross-correlation

Many of the real-world signals we are called upon to analyze are quite complex. Check out the EEG signal shown in Figure 2.2 (left side). One way to deal with these complex signals is to see how much they are like less complicated signals by using our new-found correlation tools and comparing the EEG signal with some less complicated signals such as sinusoids. Comparing with sinusoids has the advantage as it is easy to interpret the results; sinusoids represent oscillatory behavior so, when we compare a signal with a sinusoid at a given frequency, we are actually searching for oscillatory behavior at that frequency. But the lack of correlation between sine and cosine at the same frequency becomes troubling. Figure 2.12 illustrates the problem finding the correlation between a cosine and a sine with a phase shift (i.e., a sinusoid). The correlation depends on the phase shift of the sine wave. In Figure 2.12A the sine is not shifted and there is zero correlation, but when the sine is shifted by 45 degrees (Figure 2.12B) the correlation coefficient is 0.71, and at a 90-degree shift, the correlation coefficient is 1.0 (Figure 2.12C). Figure 2.12D shows that the correlation as a function of sinusoidal shift is itself a sine wave ranging between $\pm$ 1.

This complicates a search for oscillatory behavior. To test for oscillatory (i.e., sinusoidal) behavior at, say, 14 Hz, we might compare the EEG with a 14 Hz sine wave. Maybe the EEG is highly oscillatory at 14 Hz, but it is more like a cosine wave than a sine wave. The correlation with a sine wave could be very low and we would mistakenly assume there was no oscillatory behavior around 14 Hz. Okay, we could correlate the signal with both a cosine and sine, but what if the sinusoidal behavior had a phase shift that placed it halfway between a sine and cosine wave (i.e., 45 degree)?[10]

To restate the problem, when using a reference signal to search for a particular behavior we could miss it depending on the time position (or phase) of the reference signal. Figure 2.12D shows correlations as a function of time shift of the sine wave reference signal and shows that at some shift (0.125 s in this particular situation) it is an exact match. Whenever we search a target signal for a particular behavior using a reference signal, we could try correlation at a bunch of different time shifts and take the maximum correlation as representing the true similarity between reference and signal. If we are to apply this shifting/correlation approach, we need to decide how much to shift the reference signal between correlations. We do not want to shift the reference by so much that we pass over the shift

[10]Actually this could work because Equations 2.23 and 2.24 show that a general sinusoid, $C\cos(2\pi ft - \theta)$, can be represented by a combined sine and cosine function. But if we were using some other waveform as our basis of comparison, it might not.

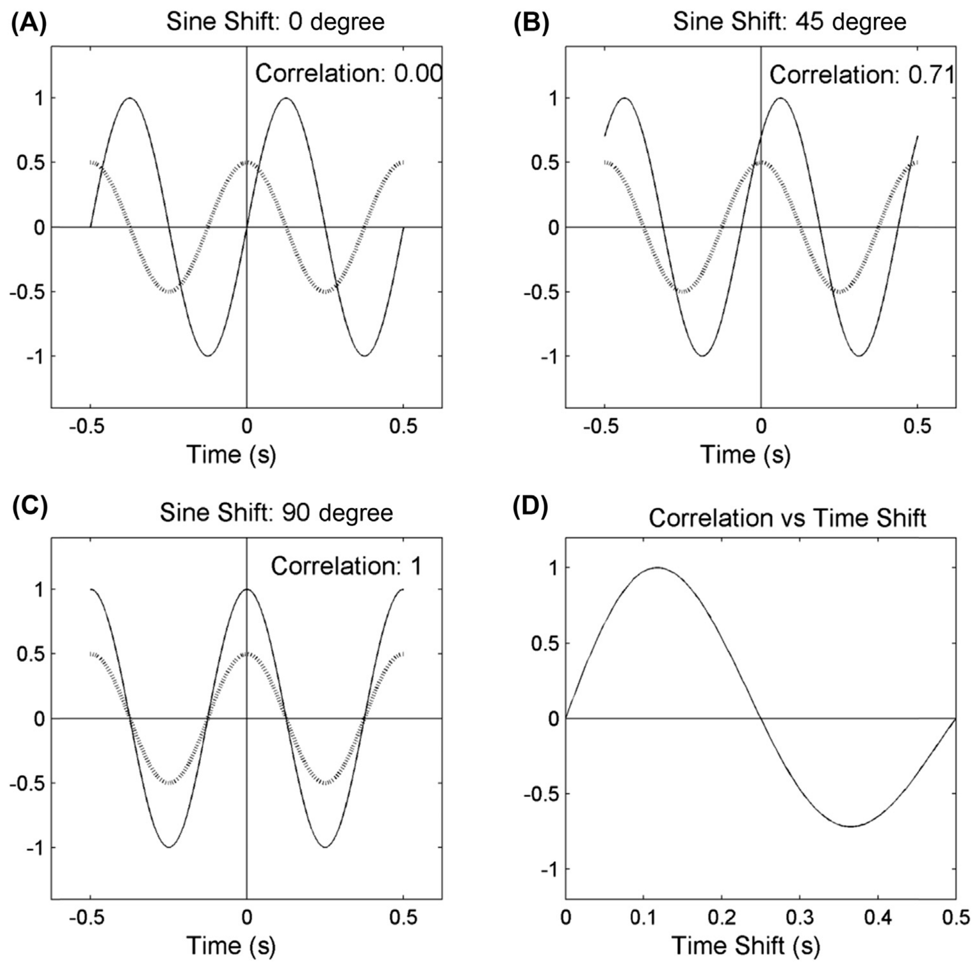


FIGURE 2.12 (A) The correlation between a 2 Hz cosine reference (dashed) and an unshifted 2.0 Hz sine wave is zero. (B) When the sine wave is time shifted by the equivalent of 45 degree, the Pearson correlation is 0.71. (C) When the sine wave is time shifted equal to 90 degree the sine wave becomes a cosine wave and the correlation is 1.0. (D) Plotting the correlation as a function of the time shift shows a sinusoidal variation. The peak value, 1.0, comes at 0.125 s, which happens to be a phase shift of 90 degrees. At a time shift of 0.25 s the sine wave is a sine wave again (but inverted) so the correlation is back to zero. At a time shift of 0.375 s the sine wave (now shifted by 270 degree) becomes an inverted cosine wave so it has a correlation of −1.0 with a cosine wave.

that gives maximum correlation between the reference and signal. If we are dealing with digital data the minimum shift would be one data point. If we made the shift just one data point we would be sure to hit the shift for maximum correlation. Of course, this means doing a lot of correlations, but it is the computer, not us, doing the work. We try this strategy out in the next example by searching for oscillatory behavior in the EEG signal at two frequencies: 6.5 and 14 Hz.

EXAMPLE 2.14

Find the correlation between the EEG signal given in Figure 2.2 (left side) and sinusoids at 6.5 and 14 Hz. The EEG data is stored in file EEG_data1.mat and the signal was sampled at 100 Hz. Is the oscillatory behavior greater at 14 Hz sinusoid or 6.5 Hz? Also generate two plots of the EEG signal superimposed on each of the two sinusoids shifted for best correlation. Scale the plots to display all the signals nicely.

Solution: We know how to perform correlation using Equation 2.29 or the unnormalized version (Equation 2.30). We are only asked to assess the relative correlation at two different frequencies, so we only need to do an unnormalized correlation.

Time shifting should be easy, just digital bookkeeping, but we need to find out what to do at the end points. This is a recurring problem in digital signal processing and we describe some common strategies in Section 2.4.4.1. Most commonly, when a signal is shifted, zeros are added to the ends as needed. We discuss this in more detail later, but here, since the reference waveform is periodic, we warp the points around: tack the end points onto the beginning. An example of this is shown in Figure 2.13 where a 2.0 Hz cosine wave (solid line) is shifted left (dashed line). The end points are wrapped around so it still looks like a sinusoid, just phase shifted. Such a shift is easy to do in MATLAB as shown in the code below.

```
% Example 2.14 Find the correlation between the EEG signal given in Figure 1.7 and
%  sinusoids at 6.5 and 14 Hz. Is the oscillatory behavior greater at 6.5 or 14 Hz?
%
load eeg_data1;          % Get the EEG data (in variable 'eeg')
N = length(eeg);         % Number of EEG data points
fs = 100;                % Sampling frequency of EEG data (given)
f = [6.5 14];            % Frequencies of reference signals
t = (0:N-1)/fs;          % Time vector
```

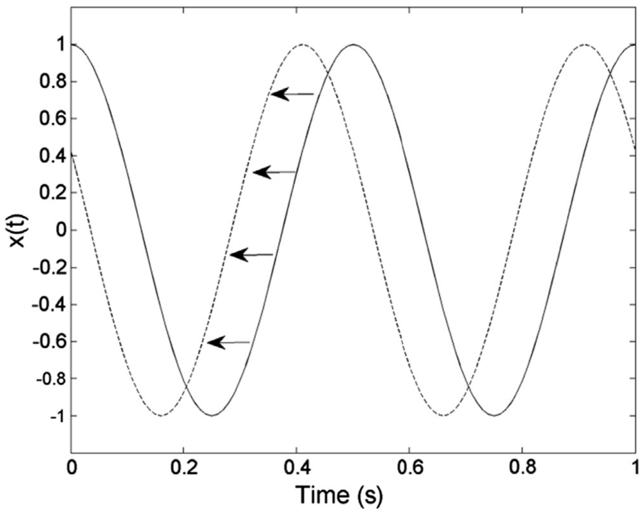


FIGURE 2.13 A 2.0 Hz cosine wave (*solid line*) is shifted by 10 data points to the left (*dashed line*).

```
for k1 = 1: 2
  x = cos(2*pi*f(k1)*t);      % Generate reference signal at desired frequency
  for k = 2:N                 % Perform N-1 correlations
    y = [x(k:end), x(1:k-1)]; % Shift reference circularly
    rxy(k) = mean(eeg.*y);   % Correlation  as a function of shift k
  end
  [corr(k1),shift(k1)] = max(rxy);   % Find maximum correlation
end
%
% Plotting section
for k = 1:2
  subplot(2,1,k);                 % Plot the two sine waves separately
  x = cos(2*pi*f(k)*t);           % Recreate the reference signal for plotting
  y = [x(shift(k):end), x(1:shift(k)-1)];   % Shift reference
  y = y * (max(eeg)/max(y))/4;    % Scale the sinusoid for good viewing
  plot(t,eeg); hold on;           % Plot EEG
  plot(t,y);                      % Plot shifted reference
  xlim([2 2.6]);                  % Scale time axis for better viewing
  ....labels and text.........
end
```

Analysis: The analysis section uses a for loop to perform the operations at the two frequencies. In each loop, a reference signal having the desired frequency is constructed to be the same length as the data. An inner `for` loop performs $N-1$ correlations between the shifted data and the EEG signal, storing the results in an array `rxy`. The wraparound shift is pretty easy to implement: a reference signal is constructed by removing the first $k-1$ points and tacking them on the end. The correlation is done as in Example 2.12. An array is constructed that contains the $N-1$ correlations where the array index indicates the shift. The maximum correlation and corresponding index are found using the `max` operator and saved.

The plotting section also uses a for loop. The reference signal is reconstructed and shifted to the position that gives the best correlation. Since the reference signal is much smaller than the EEG signal, it is scaled to be $^1/_4$ the maximum height of the EEG signal for good viewing. Both the scaled reference signal and the EEG signal are then plotted using the time vector. For better viewing the time axis is expanded to cover a period of 2.0–2.6 s. Labels and text are then added.

Results: The results produced by this program are shown in Figure 2.14. The 14 Hz sinusoidal reference has a correlation that is three times that of the 6.5 Hz reference. Figure 2.14 also shows that while the EEG signal has a lot going on, the two scaled, shifted reference signals (dashed curves) are making every effort to match the EEG signal. Perhaps sinusoids at different frequencies would better capture the oscillatory behavior of the EEG signal, but clearly no single sinusoid could represent this signal. You might think that some collection of sinusoidal references, taken together, might provide a pretty accurate representation of the EEG signal; you would be right and that is the basis of the next chapter.

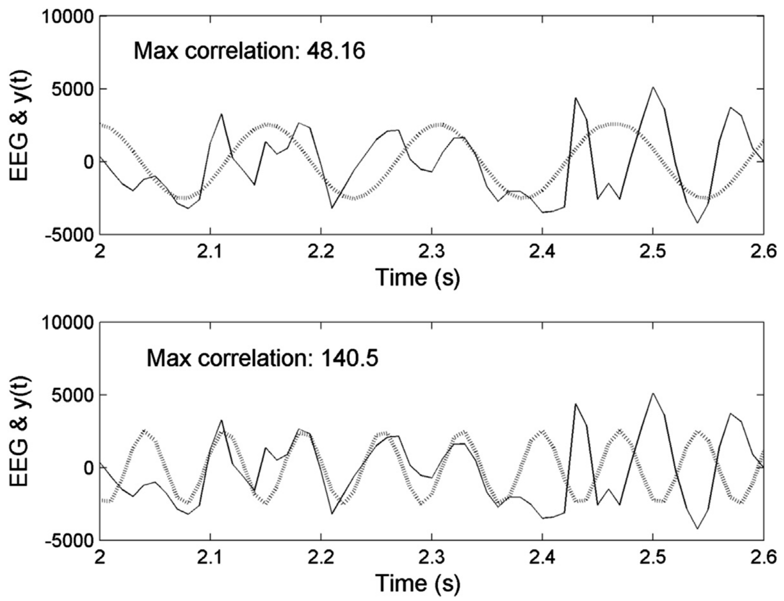


FIGURE 2.14 Plots generated by the code in Example 2.14. Both plots show a 0.6-s segment of the EEG signal shown in Figure 1.7 (*solid curves*). In the upper graph, a 6.5 Hz sinusoidal reference signal is shown shifted to best match the EEG signal (*dashed line*) and in the lower graph the shifted 14 Hz signal is shown. The shifted reference signals provide the best match possible given that they are very simple signals. These correlations show that the 14 Hz reference is a better match to the EEG signal as suggested by the signal segments shown here.

Repeated shifting and correlation of a reference signal across the entire target signal seems to work well in as much as we seem to get the best match possible. We call this approach "cross-correlation." Note that it works even if the reference signal is shorter than the target, as is sometimes the case. An equation for cross-correlation can be derived from the basic correlation equation (Equation 2.30). We just need to introduce another variable that accounts for the shift. It does not matter which function is shifted, the results would be the same. Just as in Example 2.14, the correlation operation (Equation 2.30) is performed repeatedly for different time shifts, k, and the result is a function $r_{xy}[k]$, a series of correlations for different values of k:

$$r_{xy}[k] = \frac{1}{N}\sum_{n=1}^{N} y[n]\, x[n+k] \tag{2.38}$$

where k is the shift variable[11]. To get all possible combinations between the target and reference signals, the shift variable, k, usually ranges over positive and negative integers to some max value K: $k = 0, \pm 1, \pm 2, \ldots \pm K$. This shifts the reference signal in both directions

[11]The true correlation sequence uses the estimation, or the expected value operator, applied to the product of the two signals: $y[n]$ and $x[n-k]$. However, the estimation operator requires the probability distribution functions of the two signals, which are generally not available, so applications of cross-correlation to real-world signals use Equation 2.38.

with respect to the target signal. The variable k is often called the "lags" and for the output sequence, $r_{xy}[k]$, it specifies the shift for a given correlation. If the cross-correlated signals are originally time functions, lags can be converted to time in seconds. Note the continued use of r_{xy} to indicate cross-correlation, pretty much a universal symbol for any type of correlation.

2.4.4.1 *Zero Padding*

The value of K is often equal to N (or $N - 1$), but can be less. Now we return to the end point problem. If the two signals are the same length, Figure 2.15A, as soon as we shift either signal just one position, we run out of matching points, Figure 2.15B. We need to extend the signals to create matching points. In Example 2.14 we got around this problem by using a wraparound technique: it was as if we were extending the periodic sinusoid with additional periods. With nonperiodic signals, this approach does not make as much sense. The more general approach is to extend the signals by adding zeros at the ends, Figure 2.15C. Extending a signal (or any data set) with zeros to permit calculation beyond its nominal end is called *zero padding*. This may seem like cheating, but it usually works as discussed in the next chapter.

The correlation calculation requires the two arrays be the same length, so the signals must completely overlap. To evaluate all possible alignment of the two signals, we have to extend the unshifted signal by $N - 1$ points at the beginning of the signal and another $N - 1$ points at the end of the signal. (It is $N - 1$ because we start and end with a one-point overlap.) We achieve this signal extension by zero padding as shown in Figure 2.16.

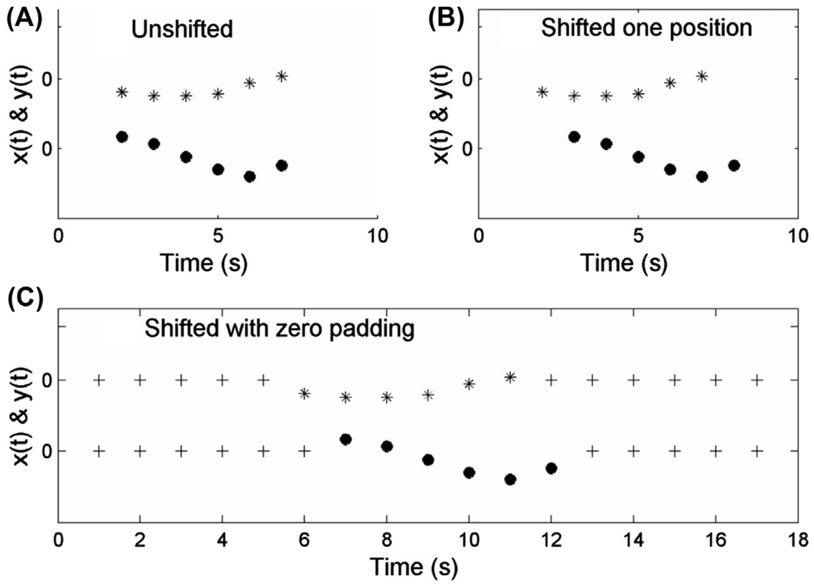


FIGURE 2.15 (A) Two signals are aligned for correlation. (B) If one of the signals (the lower one in this case) is shifted even one position as in cross-correlation, matching points are missing at both ends. (C) A common solution to this problem is to add zeros on to each end of the data, a technique known as zero padding.

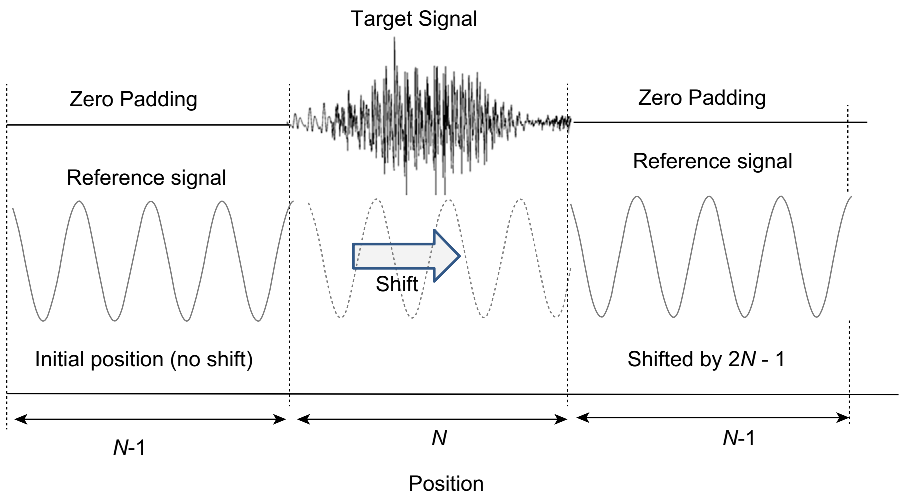


FIGURE 2.16 An example of shifting and zero padding used in cross-correlation. To cover all possible relative positions between the two signals, one signal (the sinusoid in this case) has to be shifted by $2N - 1$ positions. To have the complete overlap required by the correlation algorithm, the unshifted signal needs to be extended by $N - 1$ points at both the beginning and the end. Initially there is a one-point overlap between the reference and original, unextended signal at the inner left dashed line. After the last shift, there is a one-point overlap between the original signal and the final position of the reference signal at the right inner dashed line.

In the Signal Processing Toolbox, MATLAB features a routine that performs cross-correlation called `xcorr`. The calling structure is:

```
[rxy lags] = xcorr(x,y,maxlags);    % Perform crosscorrelation
```

where `x` and `y` are the target and reference signals (makes no difference which is which) and `maxlags` is an optional argument indicating the maximum number of shifts (the default is $2N - 1$ where N is the length of the larger input signal). The routine uses zero padding to generate the additional points. The cross-correlation found in `rxy` and `lags` is a vector the same length as `rxy` and contains the corresponding lags. This is useful in finding the lag that corresponds to a given correlation (such as the maximum correlation) or for plotting.

The `xcorr` routine has a lot of bells and whistles, including options such as various ways to bias the correlation, but we can produce a similar routine by revising the code in Example 2.14. So in the next example we modify the code in Example 2.14 and make a routine similar to `xcorr`. We use that routine to find the correlation between sinusoidal reference signals and the EEG signal, but rather than just compare at two sinusoids, let us make the comparison over a range of sinusoids from 1 to 25 Hz in 0.5-Hz increments. We also compare our routine to MATLAB's `xcorr` for these same reference signals.

EXAMPLE 2.15

Develop a routine to apply cross-correlation to a pair of signals. Perform the correlation over all possible configurations of the two signals, i.e., $2N-1$ correlations shifting one signal incrementally. Use this routine to cross-correlation the EEG signal with a collection of sinusoidal reference signals ranging from 1 to 25 Hz in 0.5-Hz increments. Plot the maximum correlation at each reference signal frequency as a function of frequency.

Solution: Correlation routine: The only modification to the code in Example 2.14 is to extend one of the signals with zeros. As a first effort, this `crosscorr` routine assumes that both signals are the same length and that the lags include all possible combinations (i.e., `maxlags` $= 2N-1$). In this routine, we arbitrarily selected signal x for padding, adding $N-1$ zeros at both the beginning and the end.

```
function [rxy lags] = crosscorr(x,y)
% Function to perform crosscorrelation similar to MATLAB's xcorr
% ......help comments describing input and output arguments .......
%
ly = length(y);                % Length of signals (both same length)
maxlags = 2*ly - 1;            % Compute maxlags from data length
x = [zeros(1,ly-1) x zeros(1,ly-1)];     % Zero pad signal x (could have been y)
for k = 1:maxlags
  x1 = x(k:k+ly-1);            % Constructed shifted signal
  rxy(k) = mean(x1.*y);        % Correlation (Equation 2.30)
  lags(k) = k - ly;            % Compute lags (useful for plotting)
end
```

Solution: Main code: Again we adopt the code from Example 2.14, replacing the cross-correlation with a call to our new routine and adjusting the frequency vector, `f`, to range from 1 to 25 in 0.5 increments. We also add a call to MATLAB's `xcorr` routine using the `"biased"` option, which uses the same scaling as our cross-correlation algorithm.

```
% Example 2.15 Find the correlation between the EEG signal and sinusoids ranging
%  from 1 to 25 Hz in 0.5-Hz increments.
%
load eeg_data1;        % This and next 3 statements identical to Example 2.13
N = length(eeg);       % Number of points
fs = 100;              % Sampling frequency of data
t = (0:N-1)/fs;        % Time vector
f = (1:0.5:25);        % Frequencies of reference signals
%
for k = 1:length(f)
  x = cos(2*pi*f(k)*t);            % Generate reference signal
  rxy = crosscorr(x,eeg);            % Compute crosscorrelation
  max_corr(k)= max(rxy);             % Find maximum correlation
  rxy_M = xcorr(x,eeg,'biased');     % Find MATLAB crosscorrelation
  max_corr_M(k) = max(rxy_M);        % MATLAB's max correlation
end
plot(f,max_corr,'k'); hold on;       % Plot crosscorr results
plot(f,max_corr_M,'*k');             % Plot MATLAB results as *-points
  ......labels.......
```

Results. Figure 2.17 shows maximum cross-correlation between the EEG signal and a sinusoidal reference signal as a function of the frequency of that reference signal. The "*" points represent the results obtained from MATLAB's `xcorr` routine and are identical to the results with `crosscorr`. The plot represents the relative amount of oscillatory behavior in the EEG signal as a function of frequency. Note a particularly active region around 8 Hz. This oscillatory activity is well known to neurophysiologists and is called the alpha wave.

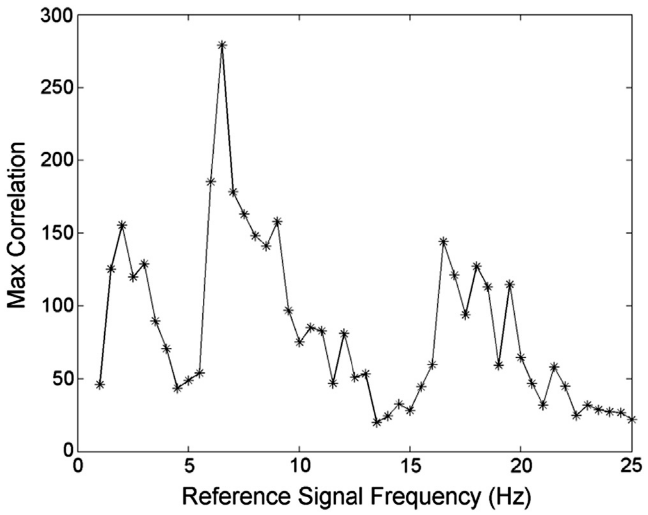


FIGURE 2.17 The plot generated by the code in Example 2.15, which shows the maximum cross-correlation between a reference sinusoid and the EEG signal of Figure 2.2 (left side). The solid line connects values obtained by the `crosscorr` routine and the "*"-points values are obtained from MATLAB's `xcorr` routine. The plot represents the amount of oscillation in the EEG signal as a function of frequency. The peak around 8 Hz is known as the alpha wave.

For continuous signals, the time shifting is continuous and the correlation becomes a continuous function of the time shift. This leads to an equation for cross-correlation that is an extension of Equation 2.31 adding a time shift variable, τ:

$$r_{xy}(\tau) = \frac{1}{T}\int_0^T y(t)x(t+\tau)dt \tag{2.39}$$

where variable τ is a continuous variable of time that specifies the time shift of $x(t)$ with respect to $y(t)$. The variable τ is analogous to the lags variable k in Equation 2.38. It is a variable of time with respect to the cross-correlation function r_{xy}, but not the time variable used to define the signals, which is denoted by the symbol "t". The τ time variable is sometimes referred to as a "dummy time variable." There is nothing particularly dumb about it, it is just a secondary time variable we use when we need two different time variables. Note that like the digital version, the continuous cross-correlation function requires multiple integrations, one integration for every value of τ. Since τ is continuous that would be an infinite number of integrations, but it is possible to evaluate Equation 2.39 analytically, at least for some very simple signals. An example of evaluating an equation similar to Equation 2.39 analytically is given in Chapter 5.

All real-world applications of cross-correlation are done in the digital domain where software such as MATLAB does all the work. It may be of some value to do at least one evaluation manually because it provides insight into the algorithm used to implement the digital cross-correlation equation, Equation 2.38.

EXAMPLE 2.16

Evaluate the cross-correlation of the two short digital signals, $x[n]$ and $y[n]$ shown in Figure 2.18 without using a computer.

Solution: To allow for all possible shifts, we need to zero pad one of the signals. Since $y[n]$ contains four samples and $x[n]$ contains three samples, we can add two zeros to each side of $y[n]$ and shift the shorter signal. The discrete cross-correlation equation (Equation 2.38) begins with one sample of $x[n]$ overlapping one sample of $y[n]$, Figure 2.19 (upper left). The calculation continues as $x[n]$ marches rightward in subsequent graphs. In this example, we use no normalization.

Results: From Figure 2.19, we can see that the unnormalized cross-correlation function is:

$$r_{xy} = [0.5, 1.10, 1.13, 0.665, 0.35, 0.125]$$

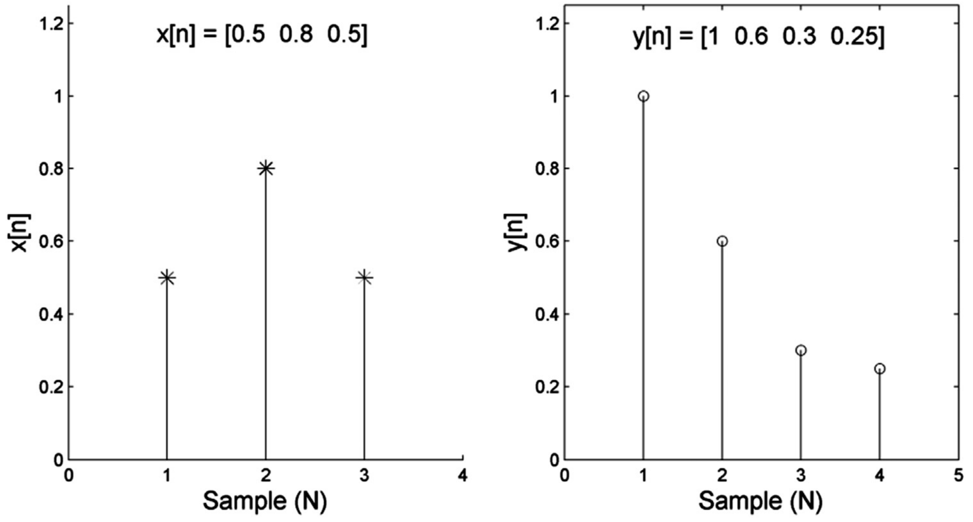


FIGURE 2.18 Two short data sequences used for manual cross-correlation shown in Example 2.16.

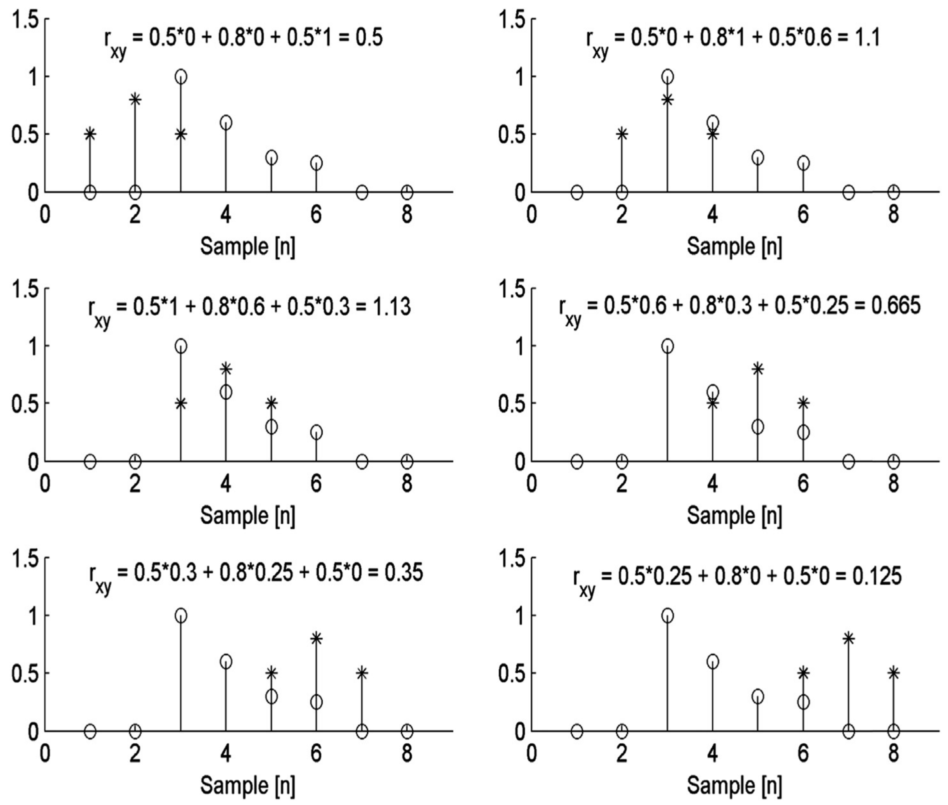


FIGURE 2.19 Results from Example 2.16, illustrating manual calculation of the cross-correlation function of the two short signals shown in Figure 2.18. The signal $x[n]$ ("*" points) is shifted across signal $y[n]$ ("o" points). The first position of $x[n]$ is at the padded zeros, two points to the left of the first valid $y[n]$ point. The highest correlation is found for a shift of three in the middle-left plot, which seems reasonable.

As in Example 2.16, cross-correlation is often used to determine the similarity between a signal and a reference waveform when the relative position for best match is unknown. Simply take the maximum value of the cross-correlation function. If needed, the shift corresponding to the maximum correlation can be determined from the lags variable. Another useful biomedical application for cross-correlation is shown in the next example: finding the time delay between two signals.

EXAMPLE 2.17

File `neural_data.mat` contains two waveforms, `x` and `y`, that were recorded from two different neurons in the brain with a sampling interval of 0.2 ms (i.e., $f_s = 5$ kHz). They are believed to be involved in the same neural operation, but are separated by one or more neuronal junctions that impart delay to one of the signals. Plot the original data, determine if they are related and, if so, the time delay between them.

Solution: Take the cross-correlation between the two signals. Find the maximum correlation and the time shift at which that maximum occurs. We use the maximum correlation to tell us if the two signals are related and the shift to determine the time delay. We can use the routine `crosscorr` developed in Example 2.15 to compute the cross-correlation, but we should scale the correlation to be between ±1 using the square root of the variances, Equation 2.32. This will better enable us to tell if the signals are related.

```
% Example 2.17 Cross-correlation of two neural signals.
%
load neural_data.mat;                  % Load data
fs = 1/0.0002;                         % Sample frequency
t = (1:length(x))/fs;                  % Calc. time vector
subplot(2,1,1);
plot(t,y,'k',t,x,':k');                % Plot data
  ........label and title.......
[rxy,lags] = crosscorr(x',y');         % Compute crosscorrelation

rxy = rxy/sqrt(var(x)*var(y));         % Scale by Pearson's; Equation 2.32

[max_corr, max_shift] = max(rxy);      % Get max correlation
time_delay = lags(max_shift)/fs;       % Delay in sec.
subplot (2,1,2);
plot(lags/fs,rxy,'k');                 % Plot crosscorrelation
  .......Labels and title.......
```

Analysis: In the call to `crosscorr` we had to transpose the two vectors since our routine assumes they are row vectors, but in fact they were column vectors. Many MATLAB routines check input vectors for orientation and adjust as needed and we modify crosscorr to do this in the next example. After cross-correlation, we find the maximum correlation using MATLAB's `max` operator. The lag corresponding to the max shift gives us the shift between the two signals in data points. To convert data points to time, as always, we multiply by T_s or divide by f_s as done here.

Result: The two signals are shown in Figure 2.20A. The dashed signal, x, follows the solid line signal, y. The cross-correlation function is seen in Figure 2.20B. Although the time at which the peak occurs is difficult to determine visually because of the large scale of the horizontal axis, the peak is found to be 0.013 s from the lag at max correlation. The max correlation is 0.45, which indicates that they are likely involved in some common neural operation.

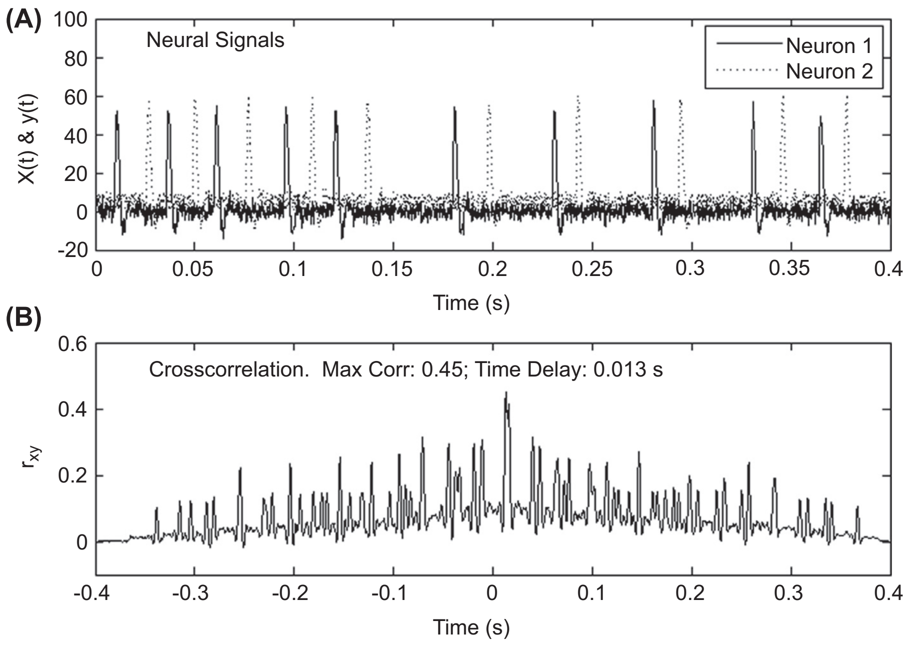


FIGURE 2.20 (A) Recordings made from two different neurons believed to be involved in the same neural operation, but delayed by an intervening synapse(s). Background noise common to such recordings is seen in both signals. (B) Cross-correlation between the two neural signals shows a peak just to the right of the center that corresponds to a shift of 0.013 s. The Pearson correlation at this point is 0.45.

2.4.4.2 *Cross-correlating Signals Having Different Lengths*

Unlike correlation, cross-correlation can be done using signals that have different lengths and is often used for probing waveforms with short reference signals. For example, given the signals seen in Figure 2.21, we might ask: "does any portion of the larger signal contain

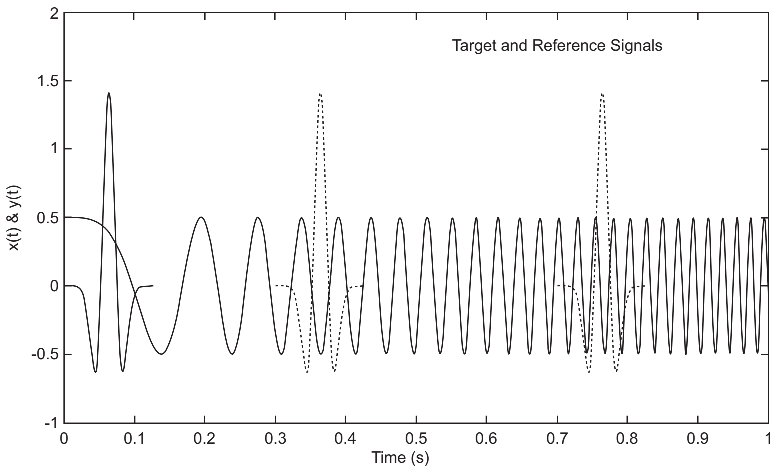


FIGURE 2.21 In Example 2.18, the short signal on the left is used as a reference signal to probe the longer target signal using cross-correlation.

something similar to the short reference signal shown on the left, and if so, how similar and where?" We answer this question using cross-correlation in our next example.

EXAMPLE 2.18

Evaluate the long signal shown in Figure 2.21 by probing it with the short signal on the left side. Determine if this target signal contains a segment that is similar to the reference, and if so, when and how much. Both signals are found as `x` (target) and `y` (reference) in file `chirp_signal.mat` and have a sampling interval of 1.0 ms.

Solution: Main Program: Follow the same procedure as in Example 2.18 except we need to modify our cross-correlation routine to allow for different signal lengths. While we are at it, we modify it to permit input signals as either row or column vectors. First we present the main code followed by the modified cross-correlation routine.

```
% Example 2.18 Cross-correlation of two signals of unequal length
%
fs = 1/.001;                              % Sampling frequency (1/Ts)
load chirp_signal;                        % Load data
[rxy,lags] = crosscorr1(x,y);             % Crosscorrelate (modified)
plot(lags/fs,rxy,'k');                    % Plot data
[max_corr max_shift] = max(rxy);          % Find max values
time_delay = lags(max_shift)/fs;          % Delay in sec.
title(['B) Max Corr: ',num2str(max_corr,2),' Time delay: ',...
  num2str(time_delay,2),' sec']);         % Put results on plot
.......label and axis.......
```

Solution: Revised Cross-correlation Routine: The revised cross-correlation routine incorporates three improvements: it can accept signal vectors as either row or column vectors, it can deal with signals of different length, and it can do a Pearson normalization if requested. The input arguments now include an optional third term: if this variable is set to "`p`," a Pearson normalization is done; if set to "`s`," a standard normalization is done (i.e., Equation 2.38.).

After checking the number of arguments and setting the default normalization if need be, the routine tests the orientation of the two input signals and transposes them if they are not already row vectors. The next section ensures that signal `y` is less than or equal to signal `x`. This makes the end point problem easier; we can always move `y` across an extended version of `x`.[12] Signal x is then zero padded at each end with the number of points in y − 1, the same as done in the previous routine. The final modification in `crosscorr1` is to apply the Pearson normalization using Equation 2.32. We now have a smarter, more flexible, easier to use cross-correlation routine.

```
function [rxy lags] = crosscorr1(x,y, normalization)
% Function to perform crosscorrelation similar to MATLAB's xcorr
% This version does not assume x and y are the same length or are row vectors
%
if nargin < 3
```

```
  normalization = 's';          % Standard normalization (1/N). Default
end
%
% Insure input signals are row vectors
[N,lx] = size(x);              % Rearrange both vectors as row vectors if needed
if N > lx                      % Rearrange as row vector
  x = x';
  lx = N;
end
[N,ly] = size(y);
if N > ly                      % Rearrange as row vector
  y = y';
  ly = N;
end
ly = length(y);                % Get new vector lengths
lx = length(x);
%
% If the input signals have unequal lengths, make y the shorter signal
if lx < ly
  temp = x;                    % Make y the shorter signal so the padded
  x = y;                       % signal will always be x
  y = temp;                    % Swap vectors x and y
end
Nx= length(x);                 % Re-establish vector lengths
Ny = length(y);
maxlags = Nx + Ny - 1;
x = [zeros(1,Ny-1) x zeros(1,Ny-1)];     % Zero pad signal x
var_y = var(y);                % Get variance of x for possible Pearson norm.
for k = 1:maxlags
  x1 = x (k:k+ly-1);           % Shift signal x

  rxy(k) = mean(x1.*y);        % Correlation (Equation 2.30)

  lags(k) = k - ly;            % Compute lags
  if normalization == 'p'      % If requested, use Pearson normalization

    rxy(k) = rxy(k)/sqrt(var(x)*var_y);   % Equation 2.32

  end
end
```

Result: The cross-correlation of the original chirp signal and the shorter reference shows a cross-correlation function that oscillates at the same frequency as the sinusoid. Note that the cross-correlation function decreases toward zero at both ends, an inevitable consequence of zero padding. (The correlations at very large and small shifts involve more zeros, Figure 2.15.) A maximum correlation of 0.66 occurs at 0.325 s, Figure 2.22 (vertical line). On comparison of the signal and reference, the time and value of max correlation looks about right.

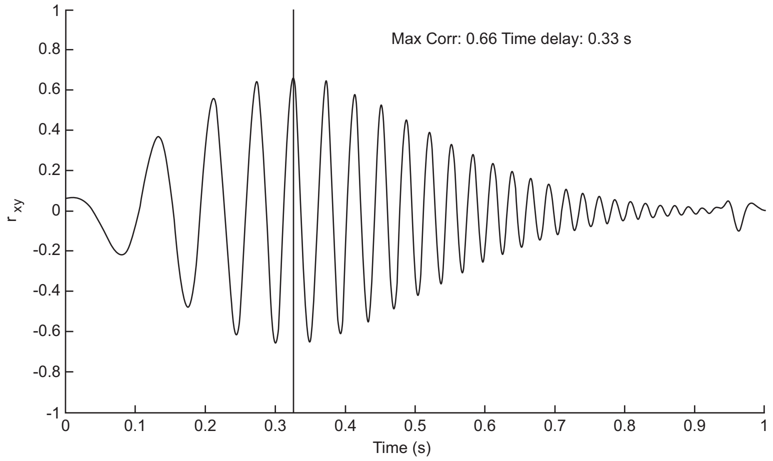


FIGURE 2.22 The cross-correlation function between the target and reference signals in Figure 2.21. The maximum Pearson correlation is moderately high at 0.66 and occurs at 0.33 s as indicated by the vertical line.

[12]The MATLAB xcorr routine simply zero pads the shorter signal to make the two equal in length. That may seem less efficient, but xcorr uses a trick to gain speed. It actually implements cross-correlation in the frequency domain. We discuss this approach in Section 5.3 on convolution because MATLAB also uses this trick to calculate convolution.

We are now quite skilled in cross-correlation, but what if we have only one signal? That is the featured topic of the next section.

2.4.5 Autocorrelation

If we have only one signal, why not try cross-correlating it with itself? Easy to do, although not so obvious what it would mean. To correlate a signal with itself, just make a copy of the single signal and apply cross-correlation (i.e., `crosscorr1(x,x)`). What does it mean? Self cross-correlation is called "autocorrelation" and basically it describes how well a signal correlates with shifted versions of itself. This could be useful in searching for repeating segments in a signal.

Autocorrelation can also be used to determine how signal data points correlate with their neighbors. As the lag (i.e., shift) increases, signal points are compared with more distant neighbors. Determining how neighboring segments relate to one another provides some insight into how the signal was generated. For example, suppose a signal remains highly correlated with itself over an extended period of time. That signal must have been produced, or modified, by a process that uses previous signal values to determine future values (at least in part). Such a process can be described as having memory, since it must remember past values of the signal to shape the signal's current values. The longer the memory, the more the signal will remain partially correlated with shifted versions of itself. Conversely, if the correlation decays after just a few lags, the system producing or acting on a signal has a short memory. If the correlation goes to zero after a shift of only one data point (i.e., lag ≥ 1), the system has no memory and that signal is actually white noise.

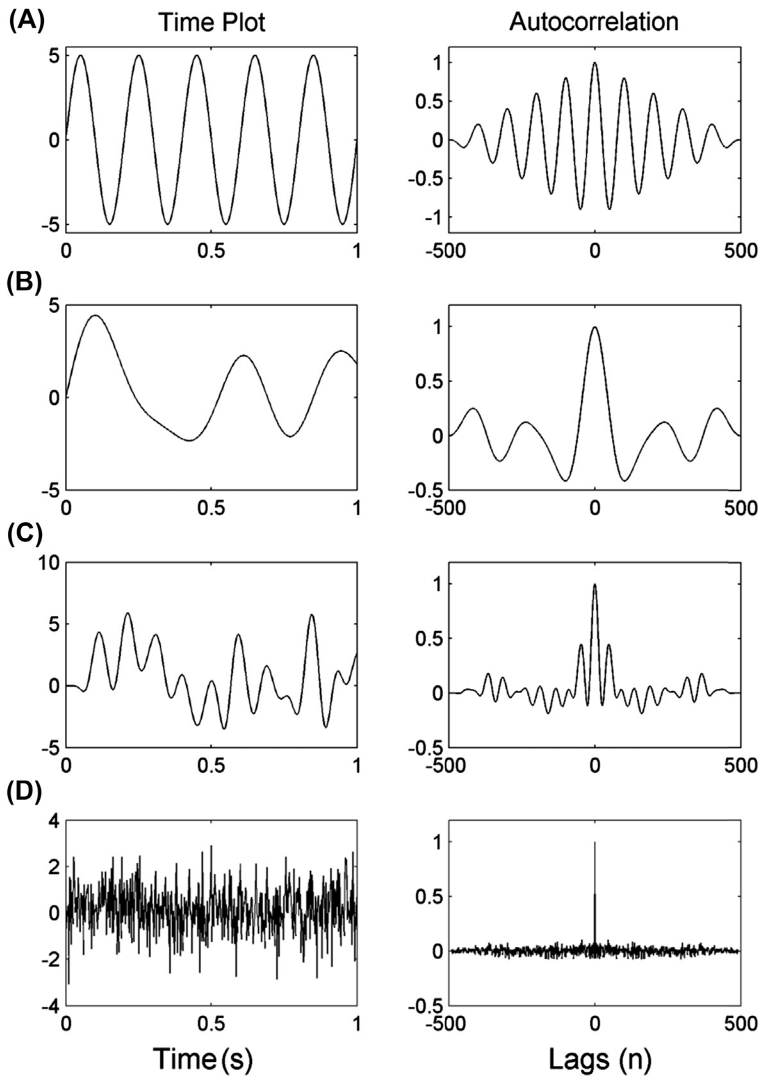


FIGURE 2.23 On the left are four different signals with their autocorrelation functions on the right. (A) A truncated sinusoid. The autocorrelation function is a cosine wave that decays due to the finite length of the signal. The autocorrelation function of an infinite sine wave would be a nondiminishing cosine wave. (B) A slowly varying signal has a slowly decaying autocorrelation function indicating that the process that acted on that signal has a relatively long memory. (C) A rapidly varying process has a rapidly decaying autocorrelation function pointing to a process with a shorter memory. (D) A random Gaussian signal has an autocorrelation function that drops to near zero for all nonzero lags.

To derive the autocorrelation equation, simply substitute the same variable for x and y in Equation 2.38 or Equation 2.39:

$$r_{xx}[k] = \frac{1}{N}\sum_{n=1}^{N} x[n]x[n+k] \tag{2.40}$$

$$r_{xx}(\tau) = \frac{1}{T}\int_0^T x(t)x(t+\tau)dt \tag{2.41}$$

where r_{xx} is the autocorrelation function and k and τ are the lag or shift variables.

Figure 2.23 shows the autocorrelation of several different waveforms. It is common to normalize the autocorrelation function to 1.0 at lag 0 (no shift) when the signal is being correlated with itself. This is the case for the autocorrelation functions in Figure 2.23. The autocorrelation of a sine wave is another cosine since the correlation varies sinusoidally with the lag. If this sine wave were infinity long the cosine would not decay, but with finite (i.e., real-world) signals, autocorrelation, like cross-correlation, requires zero padding at the signal's end points, causing the cosine function to decay at larger lags. (Again, the larger lags necessarily include more zeros in the correlation computation.)

Although all zero padded signals have an autocorrelation that decays with increasing lag,[13] the rate of that decay depends on how rapidly the signal decorrelates with itself. Decorrelation depends on how rapidly the signal fluctuates in the time domain. A rapidly varying signal, Figure 2.23C, decorrelates quickly: the average correlation between neighbors falls off rapidly with distance. You could say this signal has a poor memory of its past values and is probably the product of a process with a short memory. For slowly varying signals, the autocorrelation falls slowly, as in Figure 2.23B. For a Gaussian random signal, the correlation falls to zero instantly for all nonzero lags, both positive and negative, Figure 2.23D. This tells us that for these random signals, every data point is uncorrelated with its neighbors. Such a random signal has no memory of its past and was not operated on by a process with memory.

Since shifting the waveform with respect to itself produces the same results no matter which way the function is shifted, the autocorrelation function will be symmetrical about lag zero. Mathematically, the autocorrelation function is an even function:

$$r_{xx}(-\tau) = r_{xx}(\tau) \tag{2.42}$$

The maximum value of r_{xx} clearly occurs at zero lag, where the waveform is fully correlated with itself. If the autocorrelation is normalized by the variance that is common, the value will be one at zero lag. (Since in autocorrelation the same function is involved twice, the normalization equation given in Equation 2.32 reduces to $1/s^2$.)

When autocorrelation is implemented on a computer, it is usually considered a special case of cross-correlation. That is the case here where `crosscorr1` is used with two identical inputs.

```
[rxx, lags] = crosscorr2(x,x,'a');     % Autocorrelation
```

The `crosscor2` routine is a minor modification of `crosscorr1` constructed in Example 2.18. This modification gives the option of the common autocorrelation normalization: an `'a'` as

[13]Instead of zero padding, it is theoretically possible to extend a signal with a wraparound approach, such as that used in Example 2.14, and this might make sense if the signal was periodic. However, this would be a special case and would require modifying the standard cross-correlation routines.

the third argument normalizes the `rxx` to be 1.0 at zero shift (i.e., when `lags` = 0). A simple application of autocorrelation is shown in the next example.

EXAMPLE 2.19

Evaluate and plot the autocorrelation function of the respiratory signal `resp` in file `resp.mat` ($f_s = 125$ Hz). To better view the decrease in correlation at small shifts, plot only shifts between ±20 s.

Solution: Load the respiratory signal and use `crosscorr2` to generate the autocorrelation `function`. Plot the result against lags in seconds (i.e., divide the lags variable by f_s). Since the respiratory signal is expected to be highly oscillatory, we anticipate the autocorrelation function to be like that of a sinewave (Figure 2.23A) and to decay slowly. We scale the x axis to show only the first ±20-s lags, which should show the correlation between 6 and 12 neighbors (assuming a respiratory period of approximately 1.5—3.5 s).

```
% Example 2.19 Program to plot autocorrelation function of respiratory data
%
load resp;                              % Get data (resp)
fs = 125;                               % Sample frequency 100 Hz
[rxx,lags] =crosscorr2(resp,resp,'a');  % Autocorrelation
plot(lags/fs, rxx); hold on;            % Plot autocorrelation function
plot([lags(1) lags(end)], [0 0]);       % Plot zero line
axis([-20 20 -0.2 1.2]);                % Scale x-axis to be between ± 20 sec
       .......title and labels.......
```

Result: The autocorrelation function of the respiratory signal is shown in Figure 2.24. As expected, the signal decorrelates slowly and shows an oscillatory pattern. The period of oscillation is

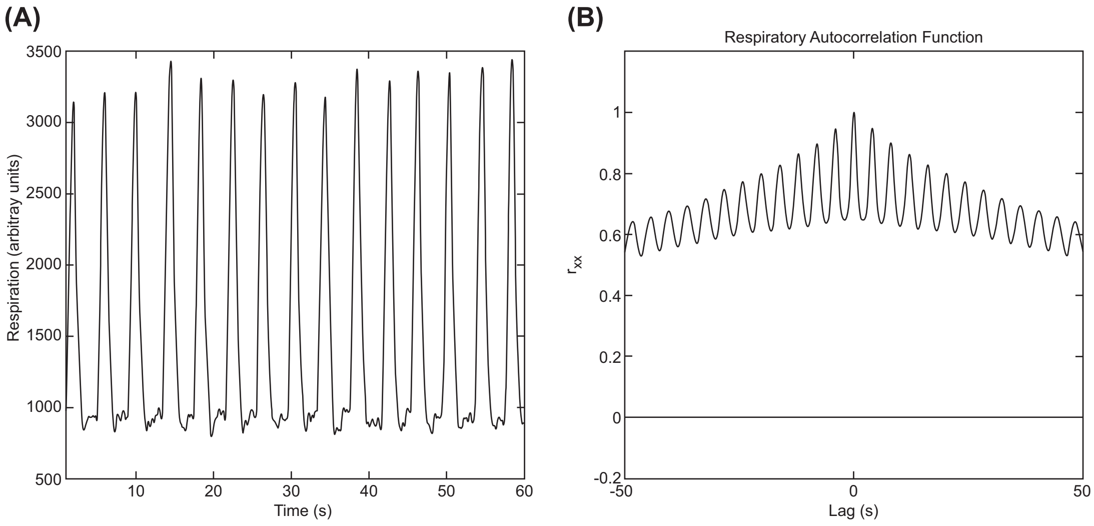


FIGURE 2.24 (A) Respiratory signal used in Example 2.19. (One minute shown.) (B) Autocorrelation function of the respiratory signal. The autocorrelation decorrelates slowly and shows oscillation. The oscillatory period of approximately 3 s reflects the subject's respiratory rate and the slow decay in correlation shows the subject had a regular breathing pattern when these data were taken.

approximately 3 s and reflects the respiratory period. (A respiratory period of 3 s corresponds to 20 breaths/min.) This example shows how autocorrelation can be used to search for oscillatory behavior in a signal. The slow decorrelation indicates that this subject had even breathing that changed very little cycle-to-cycle.

2.4.6 Autocovariance and Cross-covariance

Two operations closely related to autocorrelation and cross-correlation are autocovariance and cross-covariance. It is the same relationship we saw between correlation and covariance: in covariance operations the means are subtracted from the input signals. For cross-covariance, the discrete and continuous equations are:

$$C_{xy}[k] = \frac{1}{N}\sum_{n=1}^{N}\left(x[n]-\overline{x[n]}\right)\left(y[n+k]-\overline{y[n]}\right) \tag{2.43}$$

$$C_{xy}(\tau) = \frac{1}{T}\int_{0}^{T}\left(x(t)-\overline{x(t)}\right)\left(y(t+\tau)-\overline{y(t)}\right)dt \tag{2.44}$$

Again k ranges from 0 to $\pm K$.

For autocovariance the discrete and continuous equations become:

$$C_{xx}[k] = \frac{1}{N}\sum_{n=1}^{N}\left(x[n]-\overline{x[n]}\right)\left(x[n+k]-\overline{x[n]}\right) \tag{2.45}$$

$$C_{xx}(\tau) = \frac{1}{T}\int_{0}^{T}\left(x(t)-\overline{x(t)}\right)\left(x(t+\tau)-\overline{x(t)}\right)dt \tag{2.46}$$

The autocovariance function can be thought of as measuring the memory or self-similarity of the deviation of a signal about its mean level. Similarly, the cross-covariance function is a measure of the similarity of the deviation of two signals about their respective means. If signal means are zero, the correlation and covariance operations are identical. Only occasionally do we stumble across biosignals that have a nonzero mean. One such signal is heart rate, and analysis of heart rate variability has generated considerable interest because so many physiological processes influence the heart rate. An example of the application of autocovariance to the analysis of heart rate variability is given next.

We now understand that autocorrelation and autocovariance describe how one segment of data is correlated, on average, with adjacent segments. As mentioned previously, such correlations could be due to memory-like properties in the process that generated the data. Many physiological processes are repetitive, such as respiration and heart rate, yet vary somewhat on a cycle-to-cycle basis. Autocorrelation and cross-correlation can be

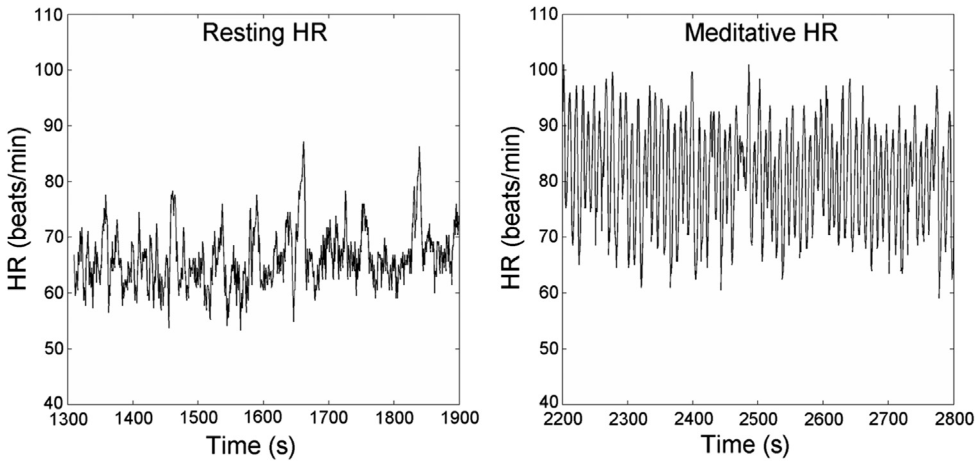


FIGURE 2.25 Ten minutes of beat-by-beat heart rate data taken from a normal resting subject and one who is meditating. Differences are substantial, with the meditative subject showing a higher overall heart rate and greater heat-to-beat fluctuations. In Example 2.20, we use autocovariance to analyze heart rate variability in the resting condition.

used to explore this variation. In the analysis of heart rate variability, autocovariance can be used to tell us if these variations are completely random or if there is some correlation between beats, or over several beats. For a variability analysis, we want to use autocovariance, not autocorrelation, since we are interested in heart rate variability, not heart rate per se. (Remember that autocovariance subtracts the mean value of the heart rate from the data and analyzes only the variation.)

Figure 2.25 shows the time plots of instantaneous heart rate in beats per minute taken under normal and meditative conditions. These data are found as `HR_pre.mat` (preliminary) and `HR_med.mat` (meditative) and are from the PhysioNet database. (Goldberger et al., 2000). Clearly the meditators have a higher average heart rate with more variability. These differences are explored in later examples, but here we look at the rate of variation and its correlation over successive beats.

EXAMPLE 2.20

Determine correlations in the heart rate variability of the resting subject whose heart rate is shown in Figure 2.25 (left side). The variability in the meditative subject is more interesting, so it is investigated in one of the problems at the end of the chapter.

Solution: Load the heart rate data taken during the two conditions. The file `Hr_pre.mat` contains the variable `hr_pre`, the instantaneous (beat-by-beat) heart rate. Since the heart rate is determined each time a heartbeat occurs, it is not evenly time-sampled and a second variable `t_pre` contains the time at which each beat is sampled. For this problem, we determine the autocovariance as a function of heart beat and we do not need the time variable. We can find the autocovariance function using `crosscorr2` as in autocorrelation as long as we first subtract the mean heart rate from the signals.

We then plot the autocovariance function and limit the x axis to ±30 successive beats to better observe the decrease in covariance with successive beats.

```
% Example 2.20 Use autocovariance to determine the correlation
%  of heart rate variation between heart beats
%
load Hr_pre;                                    % Load normal HR data
[cov_pre,lags_pre] = crosscorr2(hr_pre - mean(hr_pre),...
  hr_pre-mean(hr_pre),'a');                     % Autocovariance
plot(lags_pre,cov_pre,'k'); hold on;            % Plot resting autocovariance
plot([lags_pre(1) lags_pre(end)], [0 0],'k');   % Plot a zero line
axis([-30 30 -0.2 1.2]);                        % Limit x-axis to ± 30 beats
```

Results: The autocovariance function Figure 2.26 shows that there is some average correlation between adjacent heartbeats out to five to eight beats.

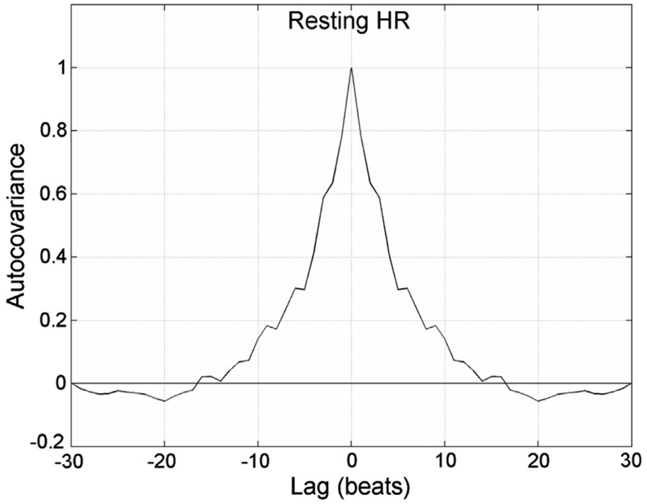


FIGURE 2.26 Autocovariance function of the heart rate variability of a normal resting subject. Some beat-to-beat correlation is seen between neighboring beats up to about five to eight beats away.

2.5 SUMMARY

The sinusoidal waveform is arguably the single most important waveform in signal processing. Some of the reasons for this importance are provided in the next chapter. Because of their importance, it is essential to know the mathematics associated with sines, cosines, and general sinusoids. Since we work with both real-valued and complex sinusoids, it is important to understand both representations and the associated math.

Some basic measurements that can be made of signals include mean values, standard deviations, variances, and rms values, all easily implemented in MATLAB. Averaging is a very powerful tool for noise reduction. If multiple observations of a physiological response can be obtained, entire responses can be averaged in an approach known as ensemble averaging. Isolating some brain activity such as its electrical response to a visual stimulus, the VER, requires averaging hundreds of responses to a repeating stimulus. Recovering the very weak evoked response that is heavily buried in the EEG is not possible without ensemble averaging.

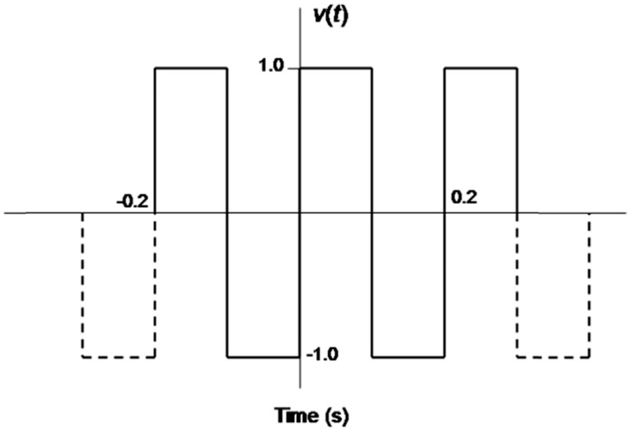
Although the basic measurements describe some fundamental signal features, they do not provide much information on signal content or meaning. A common approach to obtain more information is to probe a signal by correlating it with one or more reference waveforms. One of the most popular probing signals is the sinusoid, and sinusoidal correlation is covered in detail in the next chapter. Sometimes a signal will be correlated with another signal in its entirety, a process known as correlation (or the closely related covariance). Zero correlation between signal and reference does not necessarily mean they have nothing in common, only that the signals are mathematically orthogonal. Signals and families of signals that are orthogonal are particularly useful in signal processing because, when they are used in combination, each orthogonal signal can be treated separately: it does not interact with the other signals.

Sines and cosines have much in common; both exhibit oscillatory behavior, but they are orthogonal and their mutual correlation is zero. This presents a problem if you are probing a target signal using sinusoids: an oscillatory pattern could be missed if the target signal's oscillation was out of phase with the sinusoidal reference. A solution that works for sinusoids as well as other reference signals is to shift the reference probe so that all possible phases are correlated with the target signal. Correlation with shifting is called cross-correlation and should be used whenever we want to establish the general similarity between a reference and target signal. Cross-correlation not only quantifies the similarity between the reference and target, but also shows where the match is the greatest. Hence, cross-correlation can also be used to measure the time delay between two similar signals.

A signal can be correlated with shifted versions of itself, a process known as autocorrelation. The autocorrelation function describes the time period for which a signal remains partially correlated with itself, and this relates to the structure of the signal. A signal consisting of random noise decorrelates immediately, whereas a slowly varying signal will remain correlated over a longer period. Correlation, cross-correlation, and autocorrelation have related operations called covariances where the signals' baseline is removed before correlation. All correlations and covariances are easy to implement in MATLAB and a single routine handles these operations.

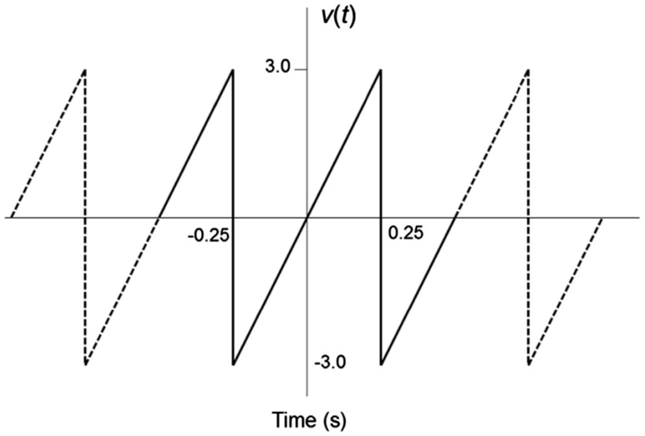
PROBLEMS

1. Use Equation 2.4 to analytically determine the rms value of a "square wave" with amplitude of 1.0 V and a period 0.2 s.



2. Generate one cycle of the square wave similar to the one shown above in a 500-point MATLAB array. Determine the RMS value of this waveform. When you take the square of the data array be sure to use a period before the up arrow so that MATLAB does the squaring point by point (i.e., `x.^2`).

3. Use Equation 2.4 to analytically determine the rms value of the waveform shown below with amplitude of 1.0 V and a period 0.5 s. (Hint: use the symmetry of the waveform to reduce the calculation required.)



4. Generate the waveform shown for Problem 3 above in MATLAB. Use 1000 points to produce one period. Take care to determine the appropriate time vector. (Constructing the function in MATLAB is more difficult than the square wave of Problem 2, but can still be done in one line of code.) Calculate the rms value of this waveform as in Problem 2. Plot this function to ensure you have constructed it correctly.
5. Fill a MATLAB vector array with 4000 Gaussianly distributed numbers (i.e., `randn`) and another with 1000 uniformly distributed numbers (i.e., `rand`). Find the mean and standard deviation of both sets of numbers. Modify the array of uniformly distributed numbers to have a mean value of zero. Confirm it has a mean of zero and recalculate the standard deviation.
6. The website file `amplitude_slice.mat` contains a signal, `x`, and an amplitude sliced version of that signal `y` ($f_s = 1$ kHz). This signal has been sliced into 16 levels. In Chapter 4, we find that amplitude slicing is like adding noise to a signal. Plot the two signals superimposed and find the rms value of the noise added to the signal by the quantization process. (Hint: subtract the sliced signal from the original and take the rms value of this difference.)
7. If a signal is measured as 2.5 V and the noise is 28 mV (28×10^{-3} V), what is the SNR in dB?
8. A single sinusoidal signal is found in a large amount of noise. (If the noise is larger than the signal, the signal is sometimes said to be "buried in noise.") If the rms value of the noise is 0.5 V and the SNR is 10 dB, what is the rms amplitude of the sinusoid?
9. The file `signal_noise.mat` contains a variable `x` that consists of a 1.0-volt peak sinusoidal signal buried in noise. What is the SNR for this signal and noise? Assume that the noise rms is much much greater than the signal rms.
10. Load the data in `ensemble_data.mat`, which contains a data matrix. The data matrix contains 100 responses of a signal in noise. In this matrix, each row is a separate response. Plot several randomly selected samples of these responses. Is it possible to identify the signal from any single record? Construct and plot the ensemble average for these data. Also construct and plot the ensemble standard deviation.
11. In this problem, we evaluate the noise reduction produced by ensemble averaging. Load the VER data variable `ver`, along with the actual, noise-free VER in variable `actual_ver`. Both these variables can be found in file `Prob2_11_data.mat`. The visual response data set consists of 1000 responses, each 100 points long. The sample interval is 5 ms. Construct an ensemble average of 25, 100, and 1000 responses. The two variables are in the correct orientation and do not have to be transposed. Subtract the noise-free variable (`actual_ver`) from an individual evoked response and the three ensemble averages to get an estimate of the noise in the three waveforms. Compute the standard deviations of the unaveraged waveform with the three averaged waveforms. Output the unaveraged standard deviation to a table of the three averaged standard deviations and the theoretical standard deviation predicted by Equation 2.17. How does this compare with the reduction predicted theoretically by Equation 2.17? Note there are practical limits to the noise reduction that you can obtain by ensemble averaging. Can you explain what might limit the continued reduction in noise as the number of responses in the average gets very large?
12. Two 10 Hz sine waves have a relative phase shift of 30 degree. What is the time difference between them? If the frequency of these sine waves doubles, but the time difference stays the same, what is the phase difference between them?

13. Convert $x(t) = -5\cos(5t) + 6\sin(5t)$ into a single sinusoid, i.e., $A\sin(5t + \theta)$.

14. Convert $x(t) = 30\sin(2t + 50)$ into sine and cosine components. (Angles should always be in degrees unless otherwise specified.)

15. Convert $x(t) = 5\cos(10t + 30) + 2\sin(10t - 20) + 6\cos(10t + 80)$ into a single sinusoid as in Problem 13.

16. Find the delay between $x_1(t) = \cos(10t + 20)$ and $x_2(t) = \sin(10t - 10)$.

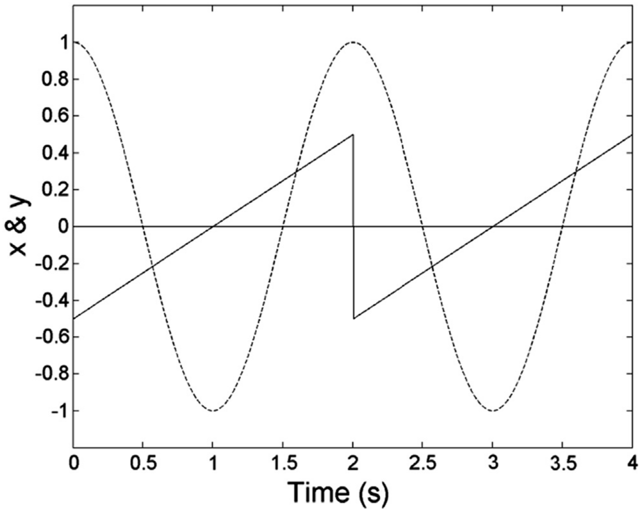
17. Equations 2.23 and 2.24 were developed to convert a sinusoid such as $\cos(\omega t - \theta)$ into a sine and cosine wave. Derive the equations to convert a sinusoid based on the sine function, $\sin(\omega t + \theta)$, into a sine and cosine wave. (Hint: Use the appropriate identity from Appendix C.)

18. (A) Given the complex number $C = -5 - j3$, find the angle using the MATLAB `atan` function. Remember MATLAB trig. functions use radians for both input and output. Note the quadrant-related error in the result. (B) Now use the `atan2` function to find the angle. Note that the arguments for this function are `atan2(b,a)`. (C) Finally, find the angle of C using the MATLAB `angle` function (i.e., `angle(C)`). Note that the angle is the same as that found by the `atan2` function. Also evaluate the magnitude of C using the MATLAB `abs` function (i.e., `abs(C)`).

19. Modify the complex exponential in Example 2.10 to generate and plot a cosine wave of amplitude 5.0 that is shifted by 45 degree. (Hint: Since the cosine is desired, you will only need to plot the real part and modify the magnitude and phase of the exponential. Remember MATLAB uses radians.)

20. Use Equation 2.31 to show that the correlation between $\sin(2\pi t)$ and $\cos(2\pi t)$ is zero. Do this both analytically and using MATLAB.

21. Use Equation 2.31 to find the correlation between the two waveforms shown below analytically.



22. Use cross-correlation to find the delay between the 10 Hz sine waves described in Problem 12. (All cross-correlation problems use MATLAB.) Use a sample frequency of

2 kHz and total time of 0.5 s. Remember MATLAB trig. functions use radians. (Note that the second part of Problem 12, finding the phase when the frequency doubled, would be difficult to do in MATLAB. There are occasions where analytical solutions are preferred.)

23. Use cross-correlation to find the time delay in seconds of the two sinusoids of Problem 16. (You choose the sampling frequency and number of data points.)

24. Use cross-correlation to find the phase shift between $x(t)$ in Problem 15 and a sine wave of the same frequency. Plot $x(t)$ and the sine wave and the cross-correlation function and find the lag at which the maximum (or minimum) correlation occurs. You choose the sample frequency and number of points. (Hint: Define $x(t)$ in MATLAB by simply writing the equation found in Problem 15 and define the second signal as $\sin(10t)$). After finding the time shift, T_d, in seconds, convert to a phase shift. (Hint: Note that the period of $x(t)$ is $T_p = 1/(10/2\pi)$ s and the phase is the ratio of the time delay to the period, times 360, i.e., $\theta = 360\frac{T_d}{T_p}$).

25. The file `two_var.mat` contains two variables `x` and `y`. Is either of these variables random? Are they orthogonal to each other? (Use any valid method to determine orthogonality.)

26. The file `prob2_26_data.mat` contains a signal `x` ($f_s = 500$ Hz). Determine if this signal contains a 50 Hz sine wave and if so at what time(s). (Hint. The `max` operator will determine only one peak. If the plot of r_{xy} suggests multiple peaks are possible, you may want to apply the `max` operator to selected segments of r_{xy}.)

27. Is the cross-correlation function of two random Gaussian variables ($N = 500$) itself a Gaussian random variable? Show. (Make your evidence definitive!)

28. The file `prob2_28_data.mat` contains a variable `x` that is primarily noise but may contain a periodic function. Plot x with the correct time axis ($f_s = 1$ kHz). Can you detect any structure in this signal? Apply autocorrelation and see if you can detect a periodic process in r_{xx}. If so, what is the frequency of this periodic process? It will help to expand the x axis to see detail. The next chapter presents a more definitive approach for detecting periodic processes and their frequencies.

29. Develop a program along the lines of Example 2.19 to determine the correlation in heart rate variability during meditation. Load file `Hr_med.mat`, which contains the heart rate in variable `hr_med` and the time vector in variable `t_med`. Calculate and plot the autocovariance. The result will show that the heart rate under meditative conditions contains some periodic elements. Can you determine the frequency of these periodic elements?

30. We know that the autocorrelation function of Gaussain random noise goes to zero for all nonzero lags, Figure 2.23D. What would you expect from a uniformly distributed random numbers? Compare the autocorrelation functions of Gaussianly and uniformly distributed random numbers. If the autocorrelation functions are different, explain.

31. This is an example of memory. Construct an array, `x`, of Gaussian random numbers (`N` = 2000). Construct a new signal from this array where each data point is a five-point running average of `x` (i.e., `y(1) = mean(x(1:5)); y(2) = mean(x(2:6)); ... y(N-5) = mean(x(N-5:N));`

Plot the autocorrelation functions of both the Gaussian and averaged signals. Expand the lag axis to ±12 lags to observe the effect of memory on the autocorrelation function.