CHAPTER

10

Stochastic, Nonstationary, and Nonlinear Systems and Signals

10.1 GOALS OF THIS CHAPTER

In this chapter, we treat some of the thornier challenges of real biological systems: nonstationarity and nonlinearity. Except for a few simulations done in Chapter 9, all of the systems we have studied thus far have been linear and time invariant (i.e., stationary) (LTI) systems. Linear systems produce linear signals, and we now have a powerful array of tools to analyze both linear signals and systems. Many linear models have been quite successful in representing biological systems even though these systems contain some nonlinearity. We also know how to use a simulation to analyze systems containing certain well-defined nonlinear processes. For example, the Stolwijk—Hardy model for extracellular glucose—insulin concentrations (Example 9.9) contains two rate-limiting nonlinearities. However, biological systems sometimes contain major nonlinearities and often change over time; i.e., they are nonstationary. The latter may be due to well-defined adaptive processes that can be added to a model, or they may be due to less well-defined degenerative processes that are hard to represent. Finally all nonlinear signals contain noise, either due to the measurement instrumentation or inherent in the signals themselves, and this makes the identification of nonlinearities particularly difficult.

Nonlinear systems produce nonlinear signals, and these nonlinear properties can provide important information about the system. Despite considerable work to date, none of the signal analysis techniques can reliably tell if a signal contains nonlinear features. Gross nonlinearities such as signal saturation can usually be detected, but identifying subtle signal nonlinearities is an ongoing challenge.

Chapters 1 and 2 introduced the basic properties of signals, including linearity versus nonlinearity, stochastic versus deterministic, and stationary versus nonstationary (see Table 2.1). Chapter 2 also introduced some of the basic measurements that can be applied to all signals in the time domain, including mean, variance, and correlations. In this chapter we expand on some of these concepts; specifically, we perform the following:

- Delve deeper into the properties of random data, including the concepts of stationarity and ergodicity.

https://doi.org/10.1016/B978-0-12-809395-5.00010-2

Copyright © 2018 Elsevier Inc. All rights reserved.

- Explore methods to analyze nonstationary signals.
- Describe ongoing work to identify and quantify subtle nonlinear signal features.
- Show how nonlinear feature detection can be applied to biological signals, particularly heart rate variability.

10.2 STOCHASTIC PROCESSES: STATIONARITY AND ERGODICITY

When dealing with signals that display stochastic (i.e., random) behavior, one of the first things we would like to know is if the signal is stationary. As briefly mentioned in Chapter 1 (Section 1.4.2), a stationary signal does not change its statistical properties over time. In other words, if we measure a statistical property, such as the mean, the value found for one segment of the data will be the same as for any other segment. If we measure the mean of a large number of segments (or an infinite number), the variance of those mean values will converge to 0. If this is the case, the signal is said to be "ergodic in the mean." For such a signal, the measurement of the mean over only one segment would be sufficient to estimate the signal's true mean (also known as the "expectation" of the mean).

The concept of ergodicity extends to other common statistical measurements including higher orders such as variance. For a system to be considered completely ergodic (without qualifiers such as "in the mean" or "in variance"), both the mean and the autocorrelation must be the same for all signal segments, or, alternatively, the first four moments of the signal. The first four moments are the mean, variance, "skewness," and "kurtosis." The mean and variance are defined in Chapter 2, and skewness and kurtosis are defined in Equations 10.1 and 10.2. If a signal satisfies these conditions, it greatly simplifies the analysis of its statistical properties because statistical measurements need to be made on only one segment of the waveform. Ergodic signals are necessarily stationary. If a signal is ergodic in the mean, it is sometimes called "weakly stationary," and in many engineering cases these signals are also ergodic. Unfortunately, bioengineering signals such as the EEG can be ergodic in the mean but still nonstationary.

To determine if a signal is ergodic, multiple measurements must be made over different time periods, and this usually requires a large amount of data. Example 10.1 compares multiple measurements of mean and variance on a band-limited Gaussian noise signal and an EEG signal. Gaussian noise signals are known to be ergodic, so we should get similar values for our measurements.

EXAMPLE 10.1

Evaluate the band-limited Gaussian signal found in the file `Bandlimit_gauss.mat` and the EEG signal in the file `EEG10.mat` to determine if they are ergodic. Both signals are stored as MATLAB variable `x` and have been normalized to have an overall RMS of 1.0. The effective time length of both signals is 10 min, where $f_s = 256$ Hz.

Solution: First we divide the data into a number of segments so that we can check the consistency of measurement between segments. We could either check the consistency of the mean and autocorrelation functions, or the first four moments. We have MATLAB routines for the first two moments (`mean` and `variance`) and can easily write a routine that calculates the third and fourth moments.

The equation for the third moment, skewness, is:

$$Skew = \frac{1}{\sigma^3 N}\sum_{n=0}^{N-1}(x_n - \overline{x})^3 \tag{10.1}$$

The fourth moment is called kurtosis and is determined as

$$Kurtosis = \frac{1}{\sigma^4 N}\sum_{n=0}^{N-1}(x_n - \overline{x})^4 \tag{10.2}$$

In both these equations, the standard deviation, σ, is normalized by $1/N$ instead of $1/(N-1)$. This is achieved in MATLAB by making the second argument of `standard deviation` as 1 (i.e., `std(x,1);`). These equations are implemented in function `skew_kurt(x)` as follows:

```
function [skew, kurt] = skew_kurt(x)
% Function to calculate skew and kurtosis of data x
%
N = length(x);          % Data length
x = x - mean(x);        % Remove mean
sd = std(x,1);          % Standard deviation (normalized by 1/N)
skew = (1/(N*sd^3))*mean(x.^3);     % Calculate skew
kurt = (1/(N*sd^4))*mean(x.^4);     % Calculate kurtosis
```

We then apply these statistical measures along with mean and variance to the two data sets. From the 10 min of data, we extract five segments, each being 30 s long. Segment length and the number of segments are somewhat arbitrary, but if the data are ergodic, we should get the same statistical values for all segments. If we are uncertain of the results, we can alter these parameters. This strategy leads to the following code:

```
% Example 10.1 Program to check for ergodicity of two signals.

%
load Bandlimit_gauss;       % Bandlimited Gaussian noise
% load EEG10;               % Alternate data set: 10 min EEG
fs = 254;                   % Sample frequency
N = 30 * fs;                % Segment size: 30 sec intervals
K = 5;                      % Divide signal into K, 30 sec intervals
for k = 1:K                 % Isolate and plot 5 segments of EEG
  ix = N*(k-1) + 1;
  y(k,:) = x(ix:ix+N-1);
  avg(k) = mean(y(k,:));    % Calculate mean
  va(k) = var(y(k,:));      % Second moment
  [skew(k), kurt(k)] = skew_kurt(y(k,:));  % Other two moments.
end
output = [avg' va' skew' kurt']      % Display individual segment results
variance = var(output)               % Display variance of individual results
```

TABLE 10.1 Value of First Four Moments of Five Band-Limited Gaussian Signals

Segment	Mean	Variance	Skewness	Kurtosis
1	0.0650	0.9853	0.0000	0.0004
2	0.0235	1.0109	−0.0000	0.0004
3	0.0182	0.9632	0.0000	−0.0004
4	0.0166	1.0042	−0.0000	0.0004
5	0.0223	1.0302	−0.0000	0.0004
Variance	0.0004	0.0007	0.0000	0.0000

Results: The values of the four moments and the associated variances are shown for the band-limited Gaussian signals in Table 10.1. The variance of all four moments is very close to 0 indicating that this signal is ergodic. Note that the third- and fourth-order moments are, themselves, also near 0 for the individual segments. This indicates that the probability distribution function for each segment is symmetrical as expected from Gaussian data.

Rerunning the program and loading the file EEG10.mat gives the results shown in Table 10.2. For this signal, the means of the segments are the same (and near 0) but not the variances. Therefore the EEG signal might be called ergodic in the mean, but it is not ergodic in variance. Again, the third- and fourth-order moments are near 0, showing that although the probability distribution functions are different for each segment, they are all symmetrical.

Example 10.1 shows how to test for ergodicity, but it requires multiple segments. If your signal is long enough, you can always slice it into multiple segments. Although the formality of testing for ergodicity is all very nice, often all you really need to do is to look at the signal. Figure 10.1A shows an EEG signal that is clearly nonstationary: the mean is changing and the

TABLE 10.2 Value of First Four Moments of Five EEG Signals

Segment	Mean	Variance	Skewness	Kurtosis
1	−0.0037	1.6631	0.0001	0.0014
2	0.0055	0.7442	0.0004	0.0041
3	0.0008	1.2456	0.0003	0.0040
4	−0.0026	0.4931	0.0003	0.0024
5	−0.0032	0.3694	0.0002	0.0020
Variance	0.0004	0.2933	0.0002	0.0020

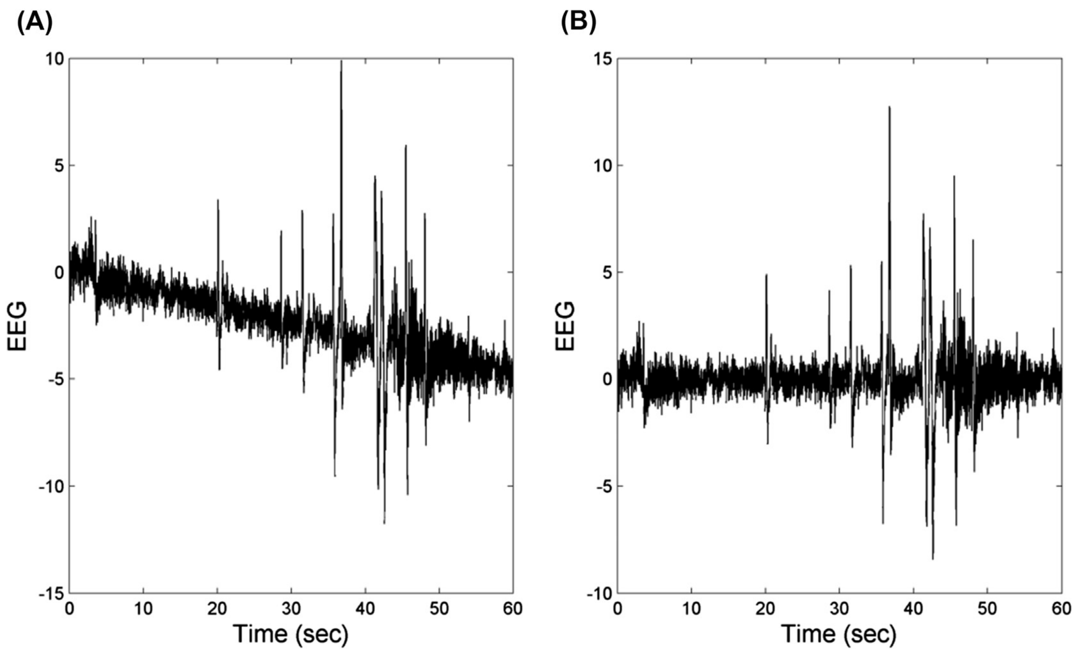

FIGURE 10.1 (A) A nonstationary EEG signal where the mean steadily decreases with time and where the variance is greater in the period between 30 and 50 s. (B) The same EEG signal where the variation in mean has been corrected using detrending, but the variation in variance remains.

variance is larger toward the end of the movement. Baseline variation can be sometimes corrected by taking the derivative, high-pass filtering, or "detrending" the data. In the later approach, baseline wander is identified and then subtracted out. The MATLAB routine `detrend(x)` subtracts out a linear, or piecewise linear, trend from the data. Detrending is described in Section 10.3.3 and the MATLAB routine is used in Example 10.9. The effect of this routine is seen in Figure 10.1B where the changing baseline has been removed. The signal is still nonstationary because the variance changes over the course of the response. Other methods for compensating for a wandering baseline are explored in the problems.

If you are unable to remove the nonstationarity, then you need to work with it. This means you have to apply analyses to short segments of the signal, segments that can be assumed to be stationary. Sometimes the nonstationarities, themselves, are of special interest. Because each signal has different features of interest, the particular analysis of a nonstationary signal is unique to each signal, but a common thread is signal segmentation. The segments may be chosen arbitrarily, but sometimes the underlying physiology suggests how the signal should be segmented. For example, in analyzing ECG signals, it is usually assumed that each beat is a stationary process but that beat-to-beat variations may be nonstationary. Thus segmenting the signal on a beat-to-beat basis makes sense. With the EEG signal, the segmentation strategy is less obvious. Nonstationarities may be due to a variety of factors including changes in mental state, but the timing of these changes may not be known in advance. In such cases, segments may be selected arbitrarily, but the appropriateness of these segments can be confirmed during the analysis. The next two examples illustrate both these strategies of signal segmentation.

The next example of signal segmentation again involves the EEG signal, a signal usually interpreted in terms of rhythmic waves in the frequency domain. The Fourier transform is used to convert the time domain EEG signal to the frequency domain. Because the EEG is nonstationary, a time−frequency analysis such as the short-term Fourier transform (STFT) is ideal (see Section 4.6 and Example 4.9). In this example, rather than apply the STFT directly, we use a band-pass filter to track the activity of a specific frequency range over time. The best known EEG rhythm is the alpha rhythm that ranges between 8 and 13 Hz. Currently these waves are thought to come from areas of the cortex that are not in use, or they may play a role in neural coordination. This example determines the alpha wave activity as it varies over a 1-h period.

EXAMPLE 10.2

Determine the time-dependent activity of a 1-h EEG signal within a frequency band of 8−13 Hz. The signal is found in the file `EEG_FP1_FP7.mat`. Segment the signal into 10-s intervals using a 50% overlap (i.e., 5 s). Then filter these isolated segments with a band-pass filter having a passband between 8 and 13 Hz. Use a fourth-order Butterworth band-pass filter. Next, take the RMS value of the filtered signal. To account for possible changes in signal gain, normalize this RMS value by the RMS value of the unfiltered EEG signal. Plot this normalized RMS value as a function of the window's time position. The one-hour EEG signal is a variable, `val`, in the file EEG and $f_s = 256$ Hz. (These data are from the PhysioBank data collecTion of physionet.org, Golberger, 2000. Record number chb03, electrode position FP1-F7.)

Solution: The analysis is straightforward. Load the data and define the parameters (f_s, filter frequencies, window size, overlap, and number of total iterations). In a loop, isolate the data using the band-pass filter, and then calculate the RMS of the filtered and unfiltered data. Store the ratio of filtered to unfiltered data and the window time. Plot the results.

```
% Example 10.2 Example to quantify the alpha activity in a 1 hr EEG signal
%
load EEG_FP1_FP7;                  % Get data
fs = 256;                          % Sample frequency (given)
N = length(val);                   % EEG length
wl = 8 *2/fs;                      % Define bandpass filter cutoff freq.
wh = 13*2/fs;
[b,a] = butter(4,[wl,wh])          % Define 4th-order bandpass filter
window_size = 10 * fs;             % 10 sec window size
overlap = 5 * fs;                  % and 5 sec overlap
incr = window_size - overlap;      % Window increment
K = round(N/incr) - 2;             % Number of windows to analyze
for k = 1:K
  i_st = incr*k;                                % Define window indices
  i_end = i_st + window_size;                   % beginning and end
  rms_wind = sqrt(mean(val(i_st:i_end).ˆ2));    % Overall rms
  alpha = filter(b,a,val(i_st:i_end));        % Filter segment
  rms(k) = sqrt(mean(alpha.^2))/rms_wind;     % Compute normalized RMS
  t(k) = mean([i_st,i_end])/fs;               % Save window center time
end
plot(t/60,rms,'k');                           % Plot results
  ......... labels..........
```

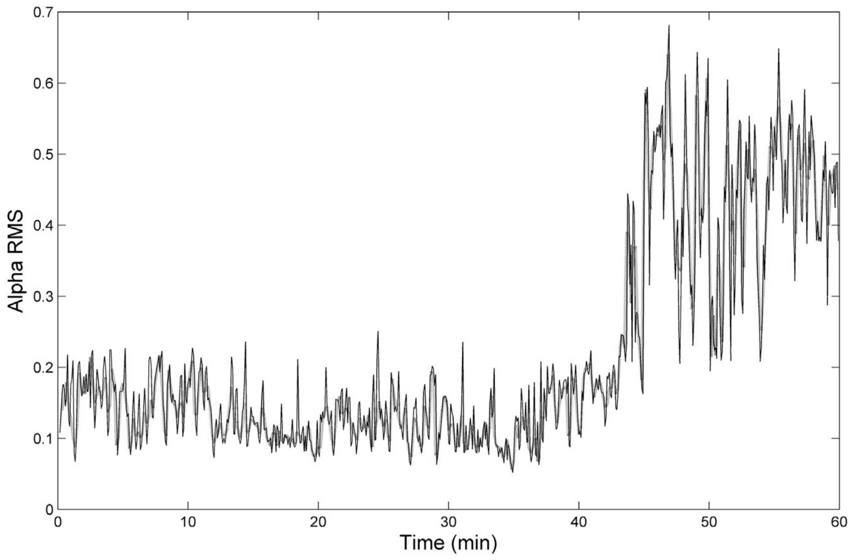

FIGURE 10.2 The alpha wave activity in a 1-h recording of EEG activity. The signal was windowed into 10-s segments with a 50% overlap. The energy of each segment between 8 and 13 Hz was determined by first isolating this frequency range with a band-pass filter and then taking the RMS signal of the result. This RMS value was normalized by the RMS of the unfiltered signal. Results obtained using a 20-s window (also 50% overlap) are superimposed over the original record as a dotted line but are difficult to see as they are quite similar indicating that exact window size is not critical.

Results: Figure 10.2 shows the alpha wave activity over the 1-h period. This activity is seen to be low initially but to increase markedly after around 45 min. We might wonder if the interval size was chosen appropriately. Perhaps a larger or smaller window would give different results. One way to check this empirically is to modify the window size and see if it changes the results. The window size here was doubled to 20 s with a 10-s overlap, and the result was plotted as a dashed line superimposed over the original results. This second line is difficult to see in Figure 10.2 because it closely follows the original results. One of the problems explores the effect of larger window sizes.

It might be interesting to see the activity of other EEG rhythms and of other electrode positions, and these are examined in the problems.

When analyzing an ECG signal, a common first step is to isolate the signal into segments that represent single cardiac cycles. Fortunately the electrical activity of the heart produces a sharp-peaked waveform known as the QRS complex. Such peaks are seen in a 4-s ECG

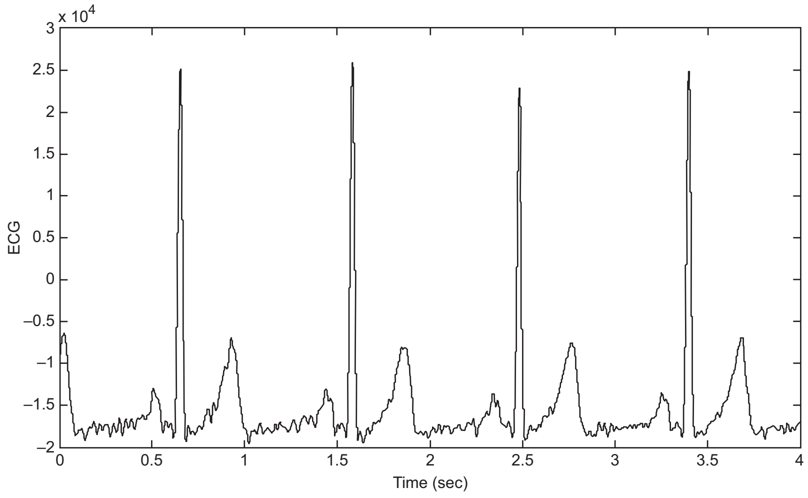

FIGURE 10.3 Portion of the 10 min ECG signal used in Example 10.3. This signal was obtained from ECG lead I.

segment in Figure 10.3. There are a number of hardware and software devices dedicated to identifying the QRS complex including the advanced Pan-Tompkins QRS detector available in MATLAB code at https://www.mathworks.com/matlabcentral/fileexchange/45840-complete-pan-tompkins-implementation-ecg-qrs-detector. However, in the next example, a simple home-brew QRS detector will be used to illustrate the basic principles.

After identifying and isolating the electrical activity associated with each beat, a typical ECG analysis might involve measuring specific features to find electrical patterns that are indicative of a disease. This step is called "feature detection" where the features are used to "classify" the beats in normal and various abnormal[1] classes. (Using features to classify data is part of a broader analysis strategy known as "pattern recognition.") Isolated abnormal electrical patterns would be further analyzed to determine their diagnostic implications. The next example gives the flavor of feature extraction and classification of beat-to-beat ECG activity.

EXAMPLE 10.3

Load the 10 min ECG signal found as variable `ECG` in the file `ECG_10min.mat` ($f_s = 100$ Hz). Detect the "R-wave," the peak of the QRS complex. Then measure two features: the mean and RMS values, of the waveform on either side of the R-wave peak. Use signal segments that begin 0.25 s before the R-wave and end 0.5 s after. Plot each feature against the other as individual points, a plot known as a "scatter plot" or "scattergram." See if you can find clusters of feature points that might indicate different types of patterns. If more than one type is found, plot examples of each waveform.

[1]Abnormal electrical patterns are associated with so-called "ectopic beats." Common examples are "premature ventricular contractions" and "premature atrial contractions". In most cases ectopic beats are benign, but some ectopic beats may indicate a greater risk of developing serious arrhythmias.

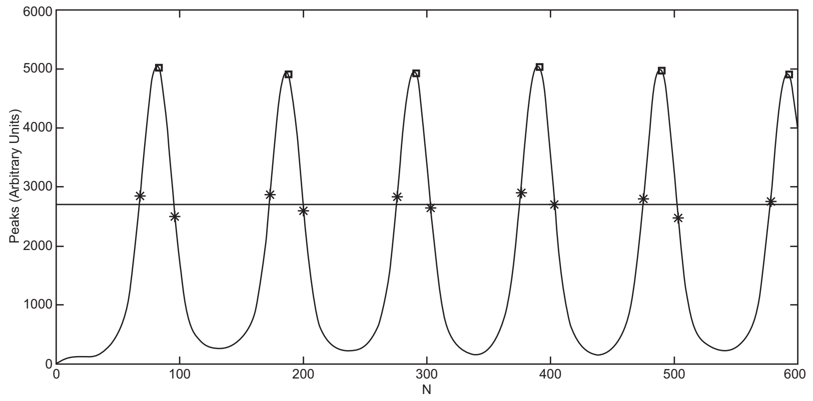

FIGURE 10.4 The curve and markers produced by the QRS detection routine `qrs_detect.m`. The smooth curve is the result of applying three filters: a band-pass filter, a derivative filter applied to the absolute value, and a moving average smoothing filter. The horizontal line is a threshold, halfway between the minimum and maximum of the smooth curve. The * points indicate where the curve crosses the threshold in the upward and downward direction and the *square points* indicate the peaks. The location of these peaks is the same as the location of the R-wave peak except for a small constant shift due to the phase characteristic of the filters.

Solution, QRS detector: Probably the most challenging component of this example is identifying the R-wave peaks. The routine `qrs_detect.m` begins with a filter strategy developed by Pan-Tompkins: a triple filter consisting of an initial band-pass filter, then a derivative filter applied to the absolute value of the signal, and finishing up with a moving average low-pass filter. (Because units are arbitrary, we do not bother to scale the derivative coefficients by $1/T_S$.) The result when applied to a typical section of ECG signal is the smooth curve shown in Figure 10.4. The rest of the routine identifies the location of the R-wave peaks. (There is a small time shift due to the phase properties of the various filters.) Finding these peaks could be done in a number of ways. Here routine `find_peaks.m` (not shown) detects when the filtered curve crosses a threshold chosen midway between the maximum and minimum values, the * points, Figure 10.4. The routine then finds the location of the maximum value between these points, shown as square points in Figure 10.4. The location of these peaks after correcting for the shift due to filtering is the output of the routine. This fairly simple approach works well on the relatively noise-free data used in this example.

```
function qrs_peaks = qrs_detect(ECG_data)
% Function to locate R-wave peaks in the ECG data
%
fs = 100;
% Initialize filter coefficients
wn = [8 16]/(fs/2);             % QRS bandpass filter freq
order = 4;                      % QRS bandpass filter order
[bn an] = butter(order,wn);     % Bandpass QRS filter coefficients
ma = ones(20,1)/20;             % Moving average filter coefficients
b = [1 0 0 0 0 0 -1];           % Derivative filter coefficients.  Skip = 3
% Triple filter ECG data
```

```
temp = filtfilt(bn,an,ECG_data);    % Bandpass filter
temp = abs(filter(b,1,temp));       % Absolute value of derivative
peaks = filtfilt(ma,1,temp);        % Moving average over 200 msec
thresh = (max(peaks) + min(peaks))/2;      % Threshold to find peaks
qrs_peaks = find_peaks(peaks,thresh);      % Find peaks using threshold crossings
qrs_peaks = qrs_peaks - 5;                 % Apply correction factor.
```

The main program isolates the activity of each beat using the R-wave peak and the time parameters specified: 0.25 s before and 0.5 s after the R-wave peak. Then the two features, RMS and mean, are easily determined for each isolated segment and stored. After all the beats are analyzed, the two features are displayed, one against the other. This plot is examined for clusters of feature points indicating qualitatively different behaviors.

```
% Example to demonstrate use and analysis of ECG signals
%
load ECG_10min;                   % Load data
fs = 100;                         % Sampling frequency
N = length(ECG);                  % Data length
qrs = qrs_detect(ECG);            % Detect QRS peaks
N_qrs = length(qrs);              % Number of QRS peaks in data
%
% Isolate single beats and measure two features
P_wave = round(0.25*fs);          % P wave sample range: 0.25 sec
qrst_wave= round(0.5*fs);         % QRS-T sample range: 0.5 sec
for k = 1:N_qrs
  ix = qrs(k);                    % Get peak location
  i1 = max([1,ix_wave]);          % Limit indices to be
  i2 = min([N,ix+qrst_wave]);     % between 1 - N
  y(k,:) = ECG(i1:i2);            % Isolate a beat
  f1(k) = sqrt(mean(y(k,:).ˆ2)); % Feature 1; RMS
  f2(k) = mean(y(k,:));           % Feature 2; mean
end
plot(f1,f2,'k*');                 % Plot features
  ........label axes........
```

Results: The scatter plot of the two features (one feature of a beat plotted against the other) is shown in Figure 10.5. It appears that there are two primary classes of beats: those with relatively small RMS values and those with a wide range of RMS values and generally higher mean values. We will call the former "cluster 1" and the latter "cluster 2." It is likely that cluster 1, which includes the majority of beats, represents normal beats, whereas cluster 2 represents some types of unusual or ectopic beats.

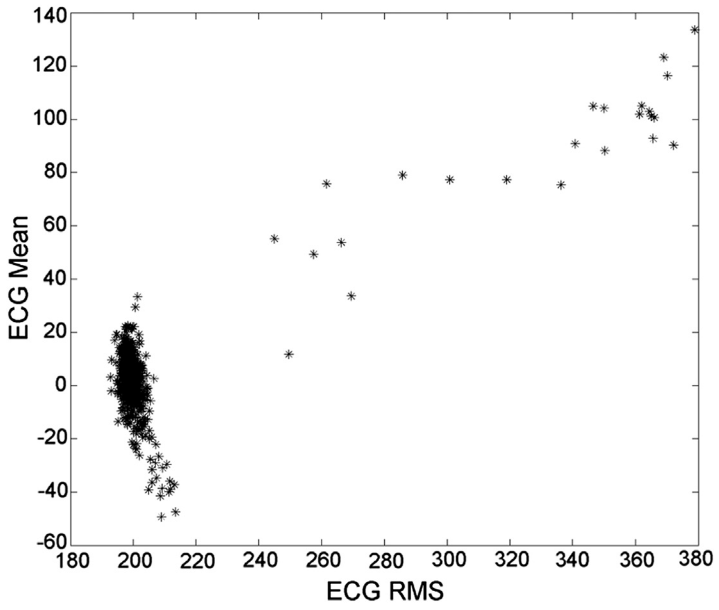

FIGURE 10.5 The scatter plot of two features measured on each beat of a 10 min ECG signal. The majority of data points cluster in the lower left corner (cluster 1), but a few are distributed across a range of higher mean and RMS values (cluster 2). Examples of beats from these two ranges are shown in Figure 10.6.

Because we saved the isolated segments, it is easy to plot some examples from the two groups. We can separate the two groups using only the RMS feature, for example, above and below an RMS of 230. The following code plots eight examples from each group superimposed using different line styles.

```
figure; hold on;                % New figure
n1 = 1;                         % Counter of group 1 plots
n2 = 1;                         % Counter of group 2 plots
t = (1:length(y(1,:)))/fs;      % Time vector for plotting
%
for k = 100:N_qrs               % Go through all records
  if f1(k) < 230 && n1 < 8      % plot only first 8 records
    plot(t,y(k,:),'k');         % Plot group 1
    n1 = n1 + 1;                % Increment group number counter
  end
  if f1(k) > 230 && n2 < 8
    plot(t,y(k,:),':k','LineWidth',2);      % Plot group 2
    n2 = n2 + 1;                % Increment group number counter
  end
end
    ........label axes........
```

The examples are shown in Figure 10.6. The examples from Cluster 1 (closely grouped lines) are very similar and follow the typical ECG pattern. Beats classified as Cluster 2 (widely varying lines) show a very different electrical pattern with a smaller QRS followed by a large, somewhat slower

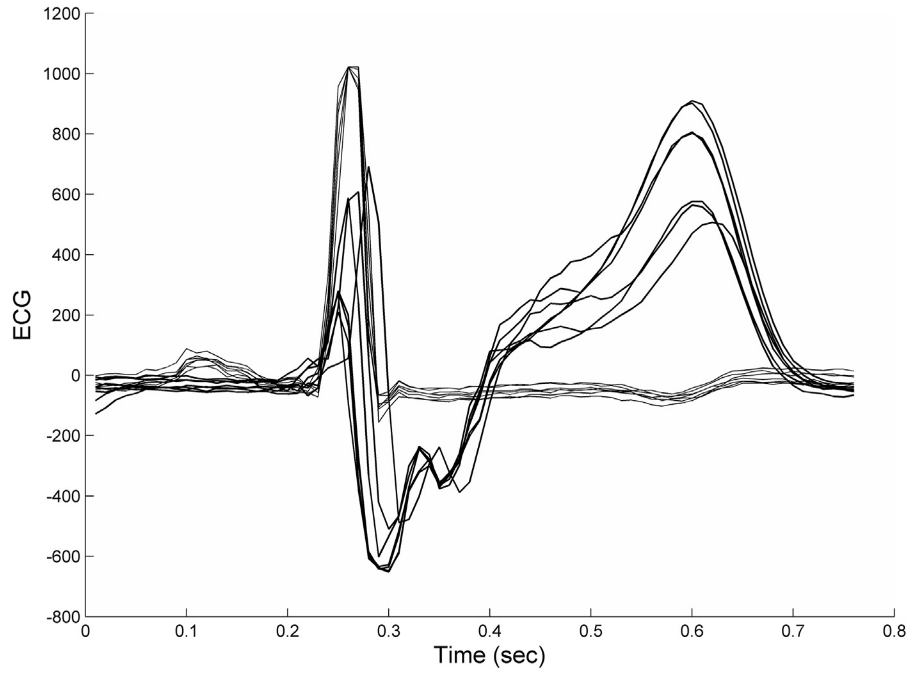

FIGURE 10.6 Examples from the two clusters found in the scatter plot of Figure 10.5. Cluster 1 samples (*closely spaced lines*) are very similar and follow a typical ECG pattern. Cluster 2 samples (*varying lines*) are more variable and show a large wave following the QRS complex.

wave. Again these unusual patterns are termed ectopic beats and are often the most diagnostically useful.

Example 10.3 touched on several important topics including QRS detection, feature extraction (a branch of pattern recognition), and classification based on feature clusters (known as "cluster analysis"). Each of these topics has been extensively developed and has made, and will continue to make, valuable contributions to biomedical engineering. Although more sophisticated approaches exist for each of these topics, Example 10.3 demonstrates the basic concepts in a real-world application. Other applications and expansion of these three topics are explored in the problems.

10.3 SIGNAL NONLINEARITY

Because nonlinear systems generate nonlinear signals and all biological systems contain some nonlinearity, you might expect all biological signals to contain some nonlinearity.

Often the nonlinearity is small and so unimportant that it can be ignored, but sometimes the most important diagnostic information is in the nonlinear behavior. Tests for nonlinearities search for two general features related to signal complexity: fractal behavior and long-term interactions. The diagnostic utility of these features drives development of new methods to identify these behaviors quickly and reliably, as they are often clouded by physiological and measurement noise.

Fractal behavior or fractal scaling methods identify similarities within a signal when viewed at different timescales or resolutions. Fractal behavior reflects the activity of multiple biological processes influencing a signal. Each process is likely to have a different timescale that leads to fractal-like behavior. For example, human heart rate is influenced by several different biological feedback loops, all having a different delay. Biological processes with long feedback loops lead to long-term signal interactions and those with short delays lead to short-term signal modifications. The health of these various physiological processes can be assessed by analyzing their impact on the signal. Fractal properties and long-term interactions have been observed in ECG and EEG signals as well as recordings of human movement such as walking gait, running gait, standing posture, and eye movements.

There are a number of popular approaches to identifying fractal behavior and long-term interaction. "Correlation dimension" and the "Lyapunov exponent" analyze a signal construct known as the "phase trajectory" in "state space" or "phase space". These approaches attempt to determine if the state space trajectory has a fractional dimension. Entropy-based methods that measure a signal's information content are used to identify both long- and short-term interactions. An approach termed "multiscale entropy" (MSE) estimates signal entropy over different timescales. These entropy estimates can be used to identify fractal scaling but can also reflect long- and short-term correlations. Another popular nonlinear method is "detrended fluctuation analysis" (DFA) that removes trends over varying timescales to differentiate long- and short-term interactions. There are an ever-increasing number of methods that search for signal nonlinearities and correlations. The three methods covered here, correlation dimension, MSE, and DFA, give a good introduction to the challenges and techniques of measuring signal complexity and nonlinearity.

10.3.1 Fractal Dimension

Fractal data show similar behavior over a wide range of scales. The classic example is a coastline that presents more or less the same random pattern over a wide range of scales, i.e., similar patterns are found when it is viewed from a high altitude, a low altitude, sea level, or even microscopically. Fractal dimension methods search for fractal behavior using a seemingly bizarre concept that a trajectory can have a fractional dimension. Although a plane is two dimensional and a sphere is three dimensional, there are trajectories that can

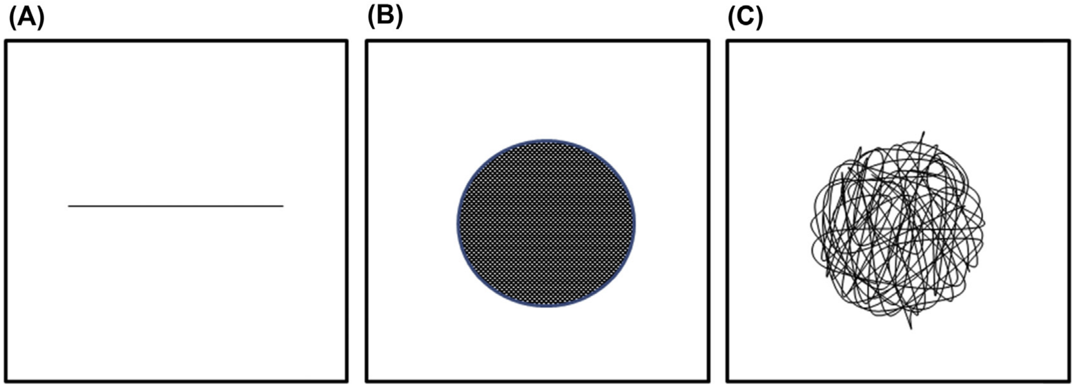

FIGURE 10.7 Three objects used to illustrate the concept of fractional dimension. (A) A one-dimensional line. (B) A two-dimensional disk. (C) A one-dimensional line but one that is so complicated that it fills almost as much space as a two-dimensional disk. To quantify this space-filling characteristic, we could extend the concept of dimension and say this object has a fractional dimension, somewhere between 1 and 2.

be, say, 2.5 dimensional. For example, the line in Figure 10.7A is a one-dimensional object (at least, in theory), whereas the disk in Figure 10.7B is clearly a two-dimensional object. But the object in Figure 10.7C is also just a line, so it is technically one dimensional, yet it has almost the appearance of a two-dimensional object (you have to use a bit of imagination here). The concept of fractional dimension is an extension of the traditional definition of dimension to include the idea of "space filling." Clearly the line in Figure 10.7C fills more space than the line in Figure 10.7A and to quantify this space-filling characteristic, we might assign the object in Figure 10.7C a fractional dimension, say, somewhere between 1 and 2.

Given this concept of fractional or fractal dimension, the question becomes what sort of objects would benefit from quantifying their space-filling characteristics. This brings us to the next new concepts of state variables and state space.

10.3.1.1 State Variables and State Space

In Chapter 5 we learned how to predict the behavior of any LTI system using convolution and the impulse response. "State variables" provide an alternative time-domain description of system behavior. State variables are internal to the system and can be used to describe internal as well as external behavior. They are more general in that they can be used with nonlinear systems and systems that are not time invariant. State variables are not unique; they are just some minimum number of variables that are required to describe the system completely. "State space," also called "phase space," is the space that defines the state variables as they move through time.

If you have a model of the system, the state variables can be derived from the model description, either based on internal model transfer functions or electrical or mechanical components as described in Chapter 12. Of course, if you have such a description of the system, you could use simulation (Chapter 9) to find the internal signals and would not need state variables. Simulation also works for systems that are neither linear nor time invariant. So although we will not actually solve for system behaviors using the state space approach, the concept is important in nonlinear signal analysis because it is the state space curves that are analyzed for their space-filling characteristics.

Although state space variables are based on a system's internal variables, they can also be found from external signals. Because the state variables are not unique, all we need is a set of variables that completely describe the system. In general, when we use external signals, the state space variables we come up with are not the same as those we would find internally. The important point is only that they relate to, in some way, the internal signals. This is the key to nonlinear signal analysis using state space concepts: if there is some internal nonlinearity whose presence is buried in the output signal, we might be able to find it by searching for special characteristics in the state space variables. This applies even when the state space trajectory is reconstructed solely from the output signal. The special characteristics we look for are fractal dimensions in the system's state space.

If the system is second order (or less), the state space variables can be obtained directly from the output signal: they are the output signal itself and its first derivative. In this case, the state space, or phase space, is two dimensional and is called the "phase plane." It is easy to plot the phase plane for any second-order system as shown in the next example.

EXAMPLE 10.4

Plot the time course and phase plane of a second-order system having the transfer function $\frac{100}{s^2+6s+100}$ in response to a unit step response. The step response of this transfer function is

$$X(s) = \frac{1}{s}TF(s) = \frac{100}{s(s^2+6s+100)} \tag{10.4}$$

Solution: Applying the methods developed in Chapter 7, we note that from the transfer function, $\omega_{n^2} = 100$, so $\omega n = 10$, and $2\delta\omega_n = 6$, so $\delta = 6/20 = 0.3$. Since $\delta < 1$, the system is underdamped. Referring to the table of Laplace transforms, the solution to the transfer function equation for complex roots is as follows:

$$x(t) = 1 - \left[\frac{e^{-\delta\omega_n t}}{\sqrt{1-\delta^2}}\sin\left(\omega_n\sqrt{1-\delta^2}\,t + \theta\right)\right] \quad \theta = \tan^{-1}\left(\frac{\sqrt{1-\delta^2}}{\delta}\right)$$

$$x(t) = 1 - \left[\frac{e^{-3t}}{0.95}\sin(9.5\,t + \theta)\right] \quad \theta = \tan^{-1}\left(\frac{0.95}{0.3}\right)$$

We solve this equation using MATLAB and plot dx/dt versus x. Because the signal is computer generated and is noise free, we can use MATLAB's `diff` routine to take the derivative. To get the proper derivative scale, we divide by T_s (or multiply the result by f_s). (Because the output of `diff` is one sample shorter than the input, we zero-pad one sample.) For the solution, assume $f_s = 1$ kHz and solve for t from 0 to 10 s. This leads to the following code:

```
% Example 10.4 Plot the time course and phase plane of a second-order system
%
fs = 1000;                 % Sampling frequency
N = fs * 10;               % Solve for 10 sec
t = (1:N)/fs;              % Time vector
theta = atan2(.95,.3);     % Solve for theta
x = 1 - (exp(-3*t).*sin(9.5*t+theta))/0.95;    % Solve equation
dx_dt = [diff(x) 0];       % Take derivative
subplot(1,2,1);
  plot(t,x,);
  ...... labels and axis limits......
subplot(1,2,2);
  plot(x,dx_dt);
  ...... labels, zero line, and limits.......
```

Results: The time plot generated by this code is shown in Figure 10.8A and the phase plane in Figure 10.8B. The inward spiraling curve in Figure 10.8B is typical of second-order overdamped systems. Note that this curve, like all state space curves, shows how the variables evolve over time but, unlike Figure 10.8A, is not itself an explicit function of time. The spiral winds down to a point at values $dx/dt = 0.0$ and $x = 1.0$. It is as if the phase trajectory is attracted to this point. For this reason, the point (which is the final position in this case) is called an "attractor." The phase trajectory of this system will be attracted to this point regardless of the initial condition. This is a "fixed point" attractor, but attractors can also be lines, surfaces, or even regions of phase space. For some

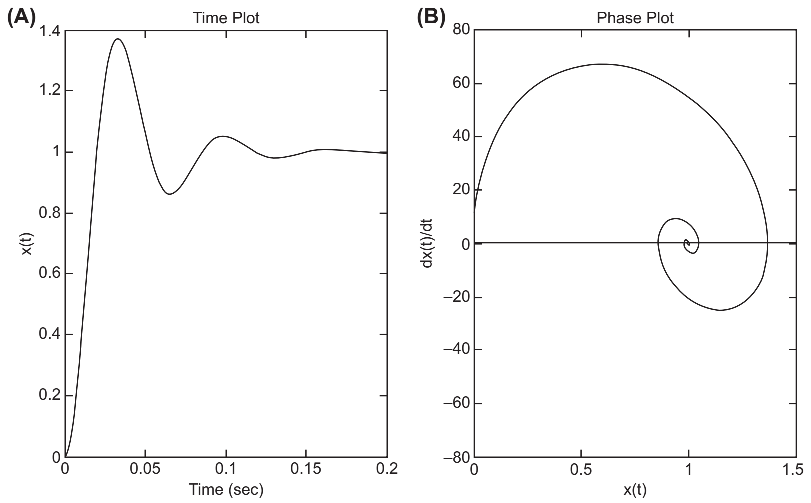

FIGURE 10.8 (A) The time plot generated by an underdamped, second-order system. (B) The phase plane produced by the time response in A. The inward spiral is typical of all such underdamped systems. The final point at (1, 0) is called the system's attractor. All trajectories for this system end up at this point, irrespective of their starting point.

systems, the phase trajectory never settles but continues to orbit; a so-called "limit cycle" attractor. An example of this is the van der Pol oscillator that was simulated in the last chapter. Its phase trajectory is explored in one of the problems. An attractor can even have a fractal structure, and it is called a "strange attractor." Strange attractors are often associated with chaotic systems. The phase trajectories developed by these attractors circle around the attractor endlessly but never repeat the same path. For an awesome demonstration of a strange attractor's phase trajectory, type `lorenz` in MATLAB.[2]

[2]Entering lorenz in MATLAB calls up an animated demo. Once the start button is pushed, the demo traces out the Lorenz phase space trajectory in real time. It has no inputs or outputs.

We now know how to find and plot the state variables of a second-order system, but what about higher-order systems? Finding the state variables of a higher-order system from only the output response uses a trick called "delay embedding" or just "embedding" and is the topic of the next section.

10.3.1.2 Delay Embedding

If all the internal state variables are coupled, they will all have an influence on the output. Delay embedding is a technique for constructing a set of likely state variables from just the output. In delay embedding, delayed versions of the measured signal are used as substitutes for the hidden internal signals. The phase space can be constructed by plotting these delayed signals against the original signal. From a single signal, a collection of signals is created, each being a delayed version of the original.

$$y(t,\tau) = [x(t); x(t-\tau); x(t-2\tau); \cdots x(t-m\tau)] \tag{10.5}$$

where τ is the constant that sets the basic delay and m is the number of additional signals created to represent the state variables. The constant m is known as the embedding dimension because the phase trajectory, which is constructed by plotting these signals against one another, now requires m dimensions.

A classic theorem by Takens states that delay embedding accurately generates the state variables, provided τ and m are appropriately chosen. Although there is some guidance in choosing these two parameters, it usually comes down to trial and error. For us, delay embedding is just a means to an end. We use it to construct a state space trajectory that is then used in conjunction with correlation dimension analysis to determine if the trajectory has a fractal dimension. A fractal dimension is a characteristic of a strange attractor and of a fractal signal. The two embedding parameters may have to be adjusted to get good results in the subsequent correlation dimension analysis.

When it comes to setting the basic delay, τ, Takens originally stated that any value would do, but with real data, there is a preferred range. If τ is too small, the constructed signals are too much alike, and if it is too large, they have no relationship to one another. Biological signals, in particular, usually have a timescale over which the samples are relevant. For example, if we assume that each beat in the ECG is independent, then a delay corresponding to several cardiac cycles would be appropriate. One method to estimate a value for τ is through the autocorrelation (Section 2.4.5), specifically, selecting the time lag where the autocorrelation function has its first minimum or reaches zero. Another approach uses mutual information, a similar function, in the same manner. Here autocorrelation is used, keeping in mind that it offers only a best guess.

Takens has come up with a rule for setting the embedding dimension, m, specifically

$$m \leq 2D + 1 \tag{10.6}$$

where m is the embedding dimension and D is the dimension of the system, in our case the system order. The problem is that we rarely know the system order or dimension. Of course we could figure that out from a mathematical description or model of the system, but if we had that we could just look at the system to see if it had any nonlinear components. Some tests have been developed such as the false nearest-neighbor analysis or single value decomposition, but these usually fail when applied to real signals. The maximum embedding dimension, m, is often limited by the length of the data, so a reasonable range of values for m can be evaluated by trial and error. The next example reconstructs the phase space of the electrical activity of the heart using the ECG signal.

EXAMPLE 10.5

Construct the phase space trajectory for the heart rate data obtained from a 30 min ECG signal. To get this signal, the peak of the R-wave was found using routine `qrs_detect` as in Example 10.3. The difference between R-wave peak indices was determined using MATLAB's `diff` routine, then divided by f_s to convert these sample differences to time intervals in seconds. The interpolation approach presented in Example 4.4 was then applied to the RR-interval data to transform it into evenly spaced time samples at 10 Hz. The heart rate signal is found as variable `hr` in the file `hr_data`. We would like to set the delay, τ, such that the delayed signals are more or less independent; a delay equivalent to three to five cardiac cycles should be sufficient. Because $f_s = 10$ Hz, we choose a delay of 40 samples (a typical cardiac cycle is around 1 s), but, in fact, a wide range of delays would work. Pragmatically, we make $m = 3$, so we can plot the trajectory on a 3D plot. In fact, previous work in heart rate analysis has suggested 3 is an appropriate value for m.

Solution: We first write a routine to perform embedding, then plot the resulting signals. The routine `delay_em.m` takes in the original signal along with constants τ and m. It outputs an array containing the delayed signals. We make the length of the output signals as long as possible, which is the input signal length minus the maximum shift, $\tau(m - 1)$.

```
function y = delay_emb(x,m,tau)
% Function to perform delay embedding on input signal x.
%
L = length(x)-(m-1)*tau;     % Length of output signals
y = zeros(m,L);              % Initialize output array for speed
for k = 1:m
  a = (1+tau*(k-1));         % Calculate first
  b = (a+L-1);               % and last indices
  y(k,1:L) = x(a:b);         % Build array in m dimensions
end
```

The main program loads the data, embeds it in a 3D space, and then plots the trajectory in 3D using MATLAB's `plot3`.

```
% Example 10.5 Program to calculate and display the phase trajectory

%  of an ECG signal
%
load hr_data;          % Get data
hr = hr-mean(hr);      % Remove mean
m = 3;                 % Define parameters
tau = 90;
y = delay_emb(x, m, tau);         % Embed signal
plot3(y(1,:), y(2,:), y(3,:));    % Plot trajectory
```

Results: The phase trajectory based on state variables reconstructed from the heart rate signal is shown in Figure 10.9. It almost fills the three dimensions showing that the order or dimension of the heart rate phase space trajectory is close to three. It is difficult to tell from this trajectory if nonlinearity is involved, but then this reconstruction is just one step in the evaluation of nonlinearity. Further analysis of the trajectory is required to determine if the system contains nonlinearities. Two popular methods that analyze the phase trajectory for nonlinear behavior are the Lyapunov exponent and correlation dimension analyses. The latter is covered in the next section.

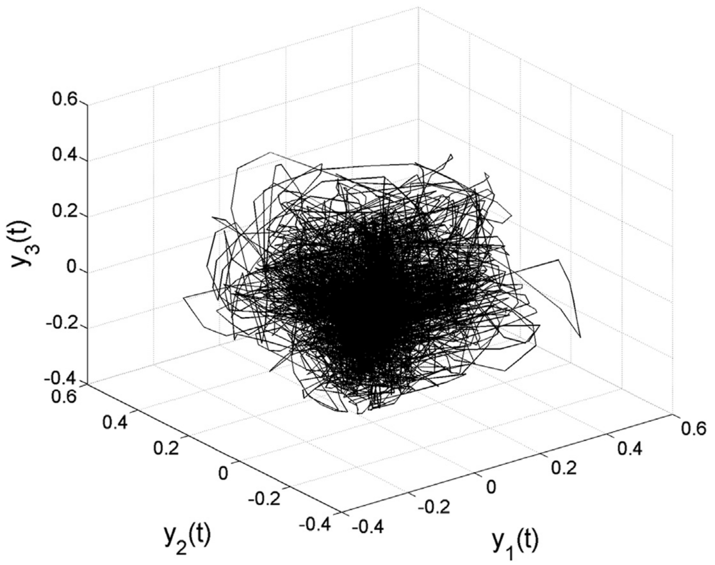

FIGURE 10.9 The phase trajectory obtained by embedding a 30 min segment of heart rate data into three dimensions. The trajectory densely fills the three dimensions.

10.3.1.3 Correlation Dimension

The correlation dimension describes the space-filling characteristics of the phase trajectory. Nonlinear chaotic systems usually have phase trajectories that fill phase space, (i.e., state space) such that the correlation dimension is fractional. A fractional correlation dimension suggests that the trajectory's attractor is fractal (i.e., a strange attractor) and that the system is chaotic.

The most popular method for estimating the correlation dimension is the correlation sum (or correlation integral if the data are continuous) that is based on the algorithm developed by Grassberger and Procaccia. Basically the Grassberger—Procaccia algorithm measures the relative relationship between the points of the phase space trajectory and describes the probability that any two points are close together. The proximity of phase space points is clearly related to how extensively the trajectory fills the space. To accomplish this measurement, the Grassberger—Procaccia algorithm measures how the number of points within a certain distance, r, varies as r increases. Specifically, it finds the distance between all two-point pairs on the trajectory and counts the number of pairs that are closer than r. The distance between any two points in a multidimensional phase space is as follows:

$$d = \sqrt{\left(\overline{x_i^2} + \overline{x_j^2}\right)} = \|x_i + x_j\| \tag{10.7}$$

where $\overline{x_i}$ and $\overline{x_j}$ are two vectors in a multidimensional space that point to the location of the two trajectory points, x_i and x_j. Measuring the distance between two points in a multidimensional space can be easily done using the matrix "norm" operator that is indicated by the double vertical bars, || ||. (Of course, MATLAB has a routine that performs the norm operation.) The distance, d, in Equation 10.7 is known as the "Euclidian distance."

To count up how many pairs of points fall within a given distance, r, we subtract the distance between two points, d, from r and count the number of subtractions that are positive using the Heaviside operator, $\Theta(x)$. The Heaviside operator is 1 for $x > 0$ and 0 otherwise. If we then take the sum over all point pairs, we get a count of the number of point pairs that were closer than r.

$$\textit{Number closer than } r = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} \Theta(r-d) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} \Theta(r - \|x[i]-x[j]\|)$$

For the Grassberger–Procaccia algorithm, we actually want the ratio of point pairs that are closer than r with respect to the total number of point pairs, so we need to normalize the aforementioned equation by $2/N(N-1)$. This gives the correlation sum equation:

$$C(r) = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} \Theta(r - \|x[i]-x[j]\|) \tag{10.8}$$

To estimate the correlation dimension from the correlation sum (Equation 10.8), we should examine how $C(r)$ changes as r gets very small. If the decrease follows a power law, i.e., $C(r) \backsim r^D$, then D is the correlation dimension. Stated mathematically, the correlation dimension, D, is defined as follows:

$$D = \lim_{r \to o} \frac{\ln C(r)}{\ln r} \tag{10.9}$$

The problem is that as r gets very small the number of point pairs decreases and the estimate is not very accurate. It would be better if more of the $C(r)$ function was involved in estimating the correlation dimension. So what is commonly done is to plot $C(r)$ against r on a log–log plot and to fit a straight line to the segment of the curve where r is small. D is then the slope of this line. Practically, Equation 10.9 is applied without the limit operator. A plot of this equation is searched for a linear region known as the "scaling region" because it determines the dimension. The next example takes a single signal, embeds it in multiple dimensions, finds the correlation sum, and then plots ln C(r) against ln r (Equation 10.9) and searches for a scaling region to estimate the correlation dimension.

EXAMPLE 10.6

Estimate the correlation dimension of the signal `x` in the file `ex10_6_data.mat`. This signal is obtained from one axis of the Lorenz phase space trajectory (see footnote 2). Because the correlation dimension analysis does not involve time, we do not need to know the sampling frequency. We can assume that the dimension of the system that produced this signal was ≤ 3.[3]

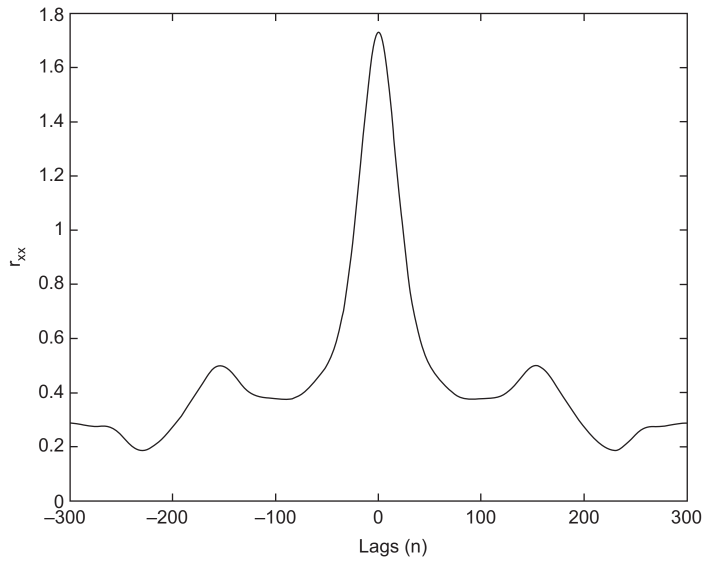

FIGURE 10.10 Autocorrelation function of the signal x before embedding. The first minimum occurs at a lags of ±100. We use that value for the delay in the delay embedding operation.

Solution: After we load the signal, the first step is to embed the signal in three dimensions using the routine `delay_emb`. To select the value for τ, we take the autocorrelation function of our signal (Figure 10.10).

The first minimum occurs at lags of ±100, so we use that value for the delay, τ, in the delay embedding routine. The original signal is embedded in three dimensions, as we are using a priori knowledge that the system is less than third order. (Otherwise we would have to try different embedding dimensions and search for values that had led to a linear scaling range in the correlation sum.)

We then plot out the phase space trajectory, as this helps us determine a range of radius values, r. The correlation sum is computationally intensive, so evaluating the sum over a large number of radius values can be quite time consuming. The phase space trajectory is shown in Figure 10.11 and

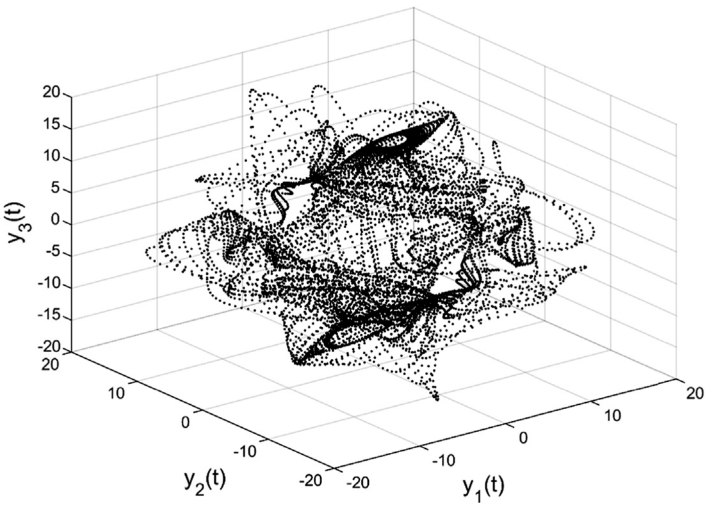

FIGURE 10.11 The phase trajectory of the signal under evaluation after being embedded in three dimensions. The trajectory ranges between ±20 in all three dimensions.

ranges between ±20 in all three dimensions. To estimate the system dimension, we need to probe values of r that are small (recall, ideally $r \rightarrow 0$). Given the scale of the phase space, we should evaluate radii much less than 1.0. Accordingly, we select a range of radii between 0.05 and 0.5.[4] If we make the increment 0.01, this leads to 46 different values for r and 46 correlation sums that require 10–15 min on a typical personal computer. The code up to this point is as follows:

```
% Example 10.6.  Program to calculate the correlation dimension of a
%  system from a single signal.
%
load ex10_6_data.mat            % Load data (Lorenz attractor)
[rxx,lags] = crosscorr(x,x);    % Calculate autocorrelation function
plot(lags,rxx);                 % and plot
xlim([-300 300]);               % Limit x-axis of plot
tau = input('Input value for tau: ');         % tau = 100, from autocorrelation
m = 3;                          % Embedding dimension (given)
y = delay_emb(x,m,tau);         % Construct the phase space trajectory
figure;
plot3(y(:,1),y(:,2),y(:,3));    % Plot trajectory
  ........labels and grid.......
r = (0.05:0.01:.5);             % Define range of r
```

The next step is to compute the correlation sum. This is done in the following routine `corr_dim`.

```
function Cr = corr_dim(y,r)
%
N = length(y(:,1));             % Number points per dimension
for k1=1:length(r)              % For each r count number of close neighbors
  for k=1:(N-1);                % For number of points count close neighbors
    repl = repmat (y(k,:),N,1);         % Copy y(k,:) into N rows
    dist = sqrt(sum((y-repl).^2,2));    % Distance between y(k,:) and all other y
    dist(1:k) = [];                     % Eliminate self-match
    nu_inside(k)= sum(dist  <  r(k1));  % Sum number of inside points
  end
  Cr(k1)=sum(nu_inside);                % Total inside points for this r
end
Cr=2*CR/(N*(N-1));                      % Normalize count
```

The correlation sum is calculated using two loops. The outer loop iterates the test radius r, whereas the inner loop totals the number of point pairs closer than r. For any given point, the distance to all other points is calculated using a MATLAB trick to improve speed. The three variables that define the test point in 3D space are duplicated into N rows using the routine `repmat`. Now the number of test points is the same as the total number of points on the trajectory, and the distance between all the point pairs can be determined in a single vectorized command. Distances less than r are counted and the inner loop repeats this calculation for all trajectory points. This strategy results in an additional self-match, which is eliminated. The counts for each point are then totaled and normalized as in Equation 10.8.

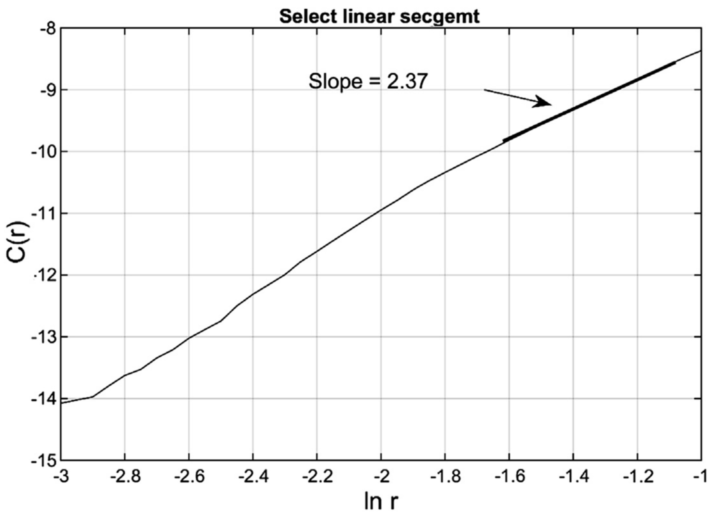

FIGURE 10.12 A plot of ln(*C*), the correlation sum versus ln(*r*), the test radius. The segment of this curve identified as most linear is shown in bold. The slope of this linear segment is an estimate of the correlation dimension: 2.37. This fractional dimension indicates that the attractor in this system is fractal and that the system is chaotic.

Returning to the main program, the ratio of ln(*C*) to ln(*r*) is plotted and examined for a scaling region, a section of the plot that is linear. A linear segment is identified interactively as a thin line superimposed on the plot. The slope of the linear region is displayed as the estimate of correlation dimension. The remaining code of Example 10.6 is as follows:

```
r = (0.05:0.01:.5);                  % Define range of r
Cr = corr_dim(y,r);                  % Calculate correlation sum
plot(log(r),log(Cr),'k'); hold on;   % Plot correlation sum
  .......labels, grid, and title......
[x,y] = ginput(2);                   % Get linear segment interactively
plot(x,y,'k','LineWidth',2);         % Plot linear segment as bold line
slope = (y(2) - y(1))/(x(2) - x(1))  % Output correlation dimension
```

Results: The plot ln(*C*) to ln(*r*) is shown in Figure 10.12. The entire curve is fairly linear, but the segment identified as most linear is shown in bold. In some of the problems, we take the derivative of ln(*C*) to help identify the flattest region. The slope of this segment is 2.37, a fractional dimension. This suggests the system has a strange attractor that is fractal and that the system behavior is chaotic. This is in the same range as that found in previous studies of this system known as the Lorenz attractor (see footnotes 2 and 3).

[3]This signal is known to be nonlinear because it is a component of the Lorenz phase space trajectory. The Lorenz phase space features a strange attractor. The Lorenz attractor is a well-studied system that is defined by three coupled first-order differential equations. The Lorenz attractor is known to have a fractional dimension between two and three as we find in this example.
[4]Because the natural log of the correlation sum is plotted against the natural log of *r* (Equation 10.9), it is common to specify *r* in terms of an exponential. For example, exp(−3:−0.5:−1) would give approximately the same range of *r* as in the current program. The approach used in this example, defining *r* as a standard sequence of numbers, is easier to appreciate.

The correlation dimension approach to identifying nonlinear behavior, specifically chaotic behavior, has a number of problems. The embedding dimension often must be chosen without much guidance. Identifying the scaling region, the linear segment of the correlation sum versus radius curve is somewhat subjective; however, this could be made more rigorous by quantifying the fit to a straight line. However, the biggest concern is that it requires relatively noise-free data, as it can easily be misled by noise. The influence of noise is illustrated in one of the problems. In the next section, we turn our attention to approaches based on information theory that attempt to estimate the entropy of a signal and how this entropy changes with scale.

10.3.2 Information-Based Methods

In this section, we examine approaches to detect nonlinearities and correlations in a signal based on the concept of entropy. In physical systems, entropy is a measure of disorder. In 1948, Claude Shannon applied this concept to signals, translating system disorder in physics into measures of signal unpredictability in signal analysis. Unpredictability is related to the information in a signal: a signal that is very regular, hence predictable, has less information than one that is unpredictable. For example, a sine wave of constant amplitude and frequency is completely predictable and carries no information.[5] Unfortunately, this relationship between entropy and information content breaks down for noise: noise is completely unpredictable, so it should have high entropy, yet like a perfectly predictable signal it carries no information.

Signal entropy has a formal definition based on probabilities, but basically entropy is related to the number of states a signal can have: signals that are very regular have a limited number of states and therefore have low entropy. When applied to signal analysis, efforts to quantify entropy search for, or describe, patterns in the signal. Here we discuss two such methods: "sample entropy" and "multiscale entropy".

10.3.2.1 Sample Entropy

Sample entropy attempts to estimate the amount of entropy, or uncertainty, of information in a signal by searching for, and measuring, repeating patterns in the signal. Sample entropy is one of the more popular methods to measure the uncertainty; it provides a single measurement based on the number of repetitions of a pattern. The basic idea is that signals that have a lot of repeated patterns are more predictable than signals that have very few repeated patterns. Sample entropy uses counts of patterns that more or less match to quantify uncertainty.

To count the number of matching patterns, we first identify a sequence of data samples to use as a "template." These template sequences are generally short, usually two or three samples. We then compare this template with all other data sequences of the same length. If the template and a sequence match, they are added to the count of matches. To count as a match, the difference between the template samples and the sequence samples must be less than some maximum error. In Figure 10.13 a two-sample template is shown that matches two other two-sample sequences in the data set.

[5]You could turn the sine wave off and on to transmit information, but then it would not have constant amplitude or frequency.

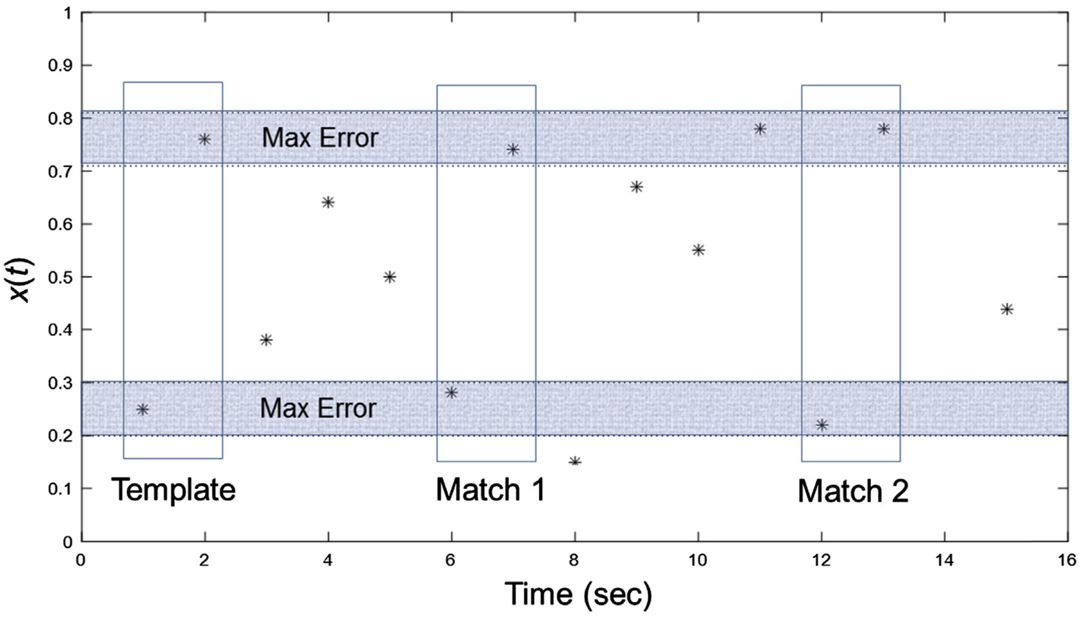

FIGURE 10.13 A two-point template on the left matches two other two-point sequences in this data set as indicated by *rectangles*. The maximum error for the data points to be considered a match in this example is ±0.05.

In the determination of sample entropy, two match counts are determined: one for a template m samples long and the other for a template $m + 1$ samples long. The sample entropy is then defined as the negative log of the ratio of $m + 1$ point matches to m point matches. It can be stated formally as follows:

$$SampEn = -\log\frac{A}{B} \tag{10.10}$$

where A is the number of $(m + 1)$-point matches and B is the number of m-point matches. Usually $m = 2$ so that A is the number of three-point matches and B is the number of two-point matches. The calculation of sample entropy is shown in the next example.

EXAMPLE 10.7

Find the sample entropy of a data set consisting of 20,000 Gaussian points

Solution: We first develop a routine to count the number of two- and three-point matches to a template. For a template, we use every two- and three-point sequence in the data set. The code for this routine is straightforward. We use an outer loop to establish an index of the template and an inner loop to index the test sequences. For each combination, we find the absolute difference between the first and second sets of samples. Because the signal is one dimensional, we can evaluate the points by subtraction and do not have to use the norm operator (i.e., || ||). When counting three-sample matches, we also test the third set of corresponding points. If all two, or three, comparisons produce an absolute difference less than the error, we increment the match counter by 1.0. A special routine is used to count 2 or 3 point matches.

```
function matches = num_matches(x,m,max_err)
% Function to determine number of matches in data set of length m
%
N = length(x);
matches = 0;         % Initialize match counter
for k = 1:N-m        % Template loop
   for j = 1:N-m     % Test point(s) loop
     if k ~= j       % Avoid self matches
       if abs(x(k) - x(j)) < max_err             % Test first points
         if abs(x(k+1) - x(j+1)) < max_err       % Test second points
           if m == 2 || abs(x(k+2) - x(j+2)) < max_err     % Test third points if
m = 3
             matches = matches + 1;              % Increment match counter
           end
         end
       end
     end
   end
  end
```

The routine `num_matches` is then used by the main program to calculate the sample entropy. After defining the random number data set, the main program calls this routine twice: once to determine two-point matches and again to determine three-point matches. The data are normalized to have a standard deviation of 1.0 and the maximum error is set to 0.15, a commonly used value.

```
% Example 10.7 Calculation of sample entropy on random data

%
N = 20000;                 % Number of data points
x = randn(1,N);            % Generate data
max_err = 0.15;            % Maximum error between points
x = (x-mean(x))/std(x);    % Subtract mean, normalize so std = 1
%
B = num_matches(x,2,max_err);          % Determine two-component matches
A = num_matches(x,3,max_err);          % Determine three-component matches
samp_entropy = -log(A/B);              % Calculate sample entropy,Equation 10.10
disp(samp_entropy)                     % and display
```

Result: The output of this program is a single number. For these data, the sample entropy is 2.4727.

Sample entropy gives us a single number that can be used to compare the information content of different signals. Changes in entropy can be of diagnostic value in certain diseases, but often of greater importance are changes in long-term processes that incorporate physiological feedback loops. To identify long- and short-term interactions, we can evaluate sample entropy over different timescales. High entropy levels suggest the influence of a process, so the variation in entropy over different timescales highlights processes that act over different time periods. Measuring entropy at different scales can also show fractal or "self-similar" behavior. Signal scaling is done essentially by "downsampling" the signal in an approach also known as "coarse graining."

10.3.2.2 Multiscale Entropy

MSE was developed to analyze the signal properties at different scales (Costa et al, 2005). In MSE, the sample entropy is determined over a range of scales, that is, at different signal resolutions. The sample entropy is then plotted as a function of scale. To view a signal or data set at different scales, MSE uses a technique called "coarse graining" to change the resolution of the signal.

10.3.2.2.1 DATA SCALING THROUGH COARSE GRAINING

Coarse graining uses downsampling, usually in conjunction with an antialiasing low-pass filter, to reduce the resolution of the signal. Reducing signal resolution illuminates processes that act over longer timescales. The finest scale is set by the resolution of the sampled signal; once it is sampled, we cannot examine the signal at a finer scale. However, we can examine the data on coarser, lower resolution scales simply by downsampling the data. Figure 10.14 shows a signal at its finest scale and at lower resolutions achieved by downsampling the original signal. Decreasing the scale enhances the slow (or low-frequency) behavior of the signal, but unlike low-pass filtering, we are not simply attenuating high frequencies; we are actually eliminating high frequency information.

Data scaling or coarse graining uses downsampling but with the addition of a low-pass filter to prevent aliasing. Recall that with discrete data, frequencies greater than $f_s/2$ are folded back and confounded (i.e., they are added in with) the lower frequencies (see Section 4.1.1), a condition known as aliasing. If the data are properly sampled, they will not contain frequencies greater than $f_s/2$. However, once we start downsampling, for example, by removing every other data point, we effectively reduce the sample frequency, so that aliasing can occur. The solution is to low-pass filter the data before downsampling.

Adding a filter before downsampling is not difficult, but the trick is to properly define the filter's cutoff frequency so as to remove potential aliasing frequencies. The cutoff frequency of the filter depends on the amount of downsampling. If we retain every other data sample, we are downsampling by a factor of two and halving the effective f_s. Setting the cutoff frequency to $f_s/4$ attenuates, at least partially, frequencies above the new $f_s/2$. If we keep every third point, we reduce f_s by 2/3, so we now want the cutoff frequency to be $f_s/6$. In general, we want the low-pass filter cutoff frequency to be $f_s/2n$, where n is the downsampling factor. Fortunately, MATLAB filter routines specify the cutoff frequency relative to $f_s/2$ (see Chapter 8 and Section 8.6), so to get the proper cutoff frequency we make $f_c = 1/n$. In the routine to perform coarse graining, we use a sixth-order Butterworth filter and, after filtering, downsample by n.

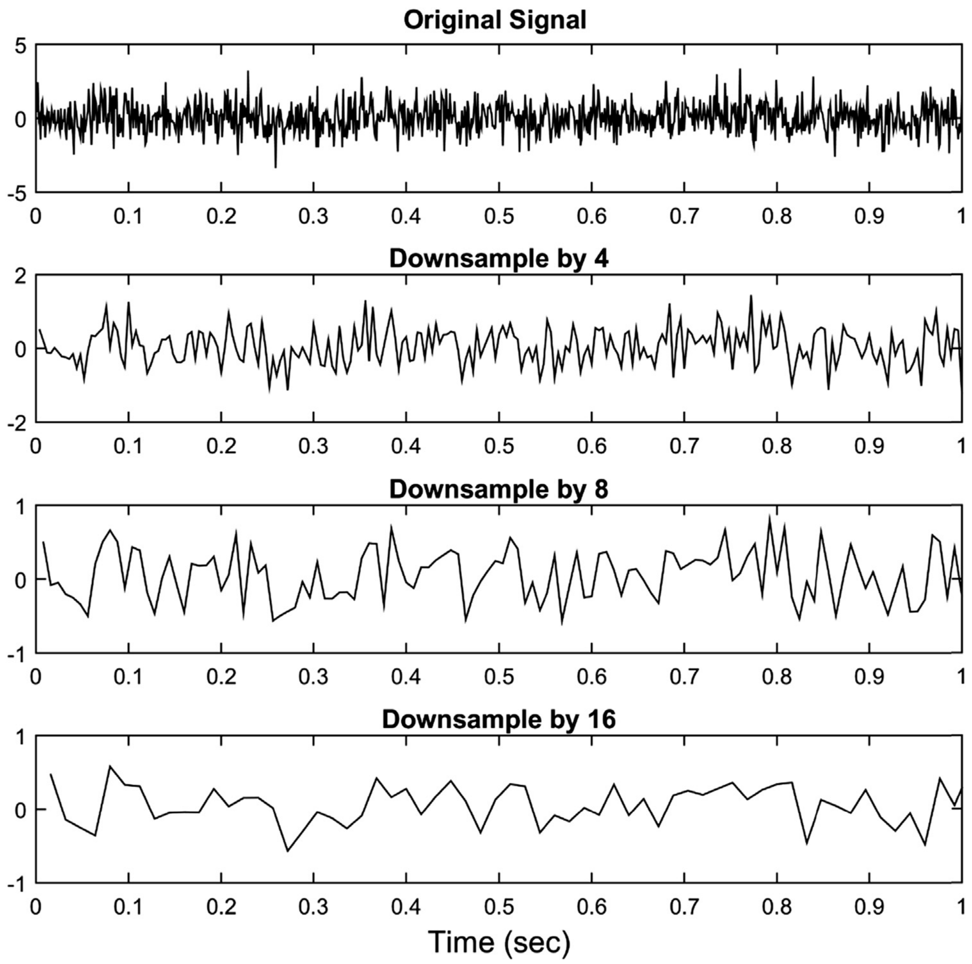

FIGURE 10.14 A signal shown at four different scales achieved by downsampling. The original signal (upper plot) has the finest resolution, whereas the downsampled signals show behavior over longer time frames. In the downsampled signals, the higher frequency information is not just attenuated; it is removed from the signal.

```
function y = coarse_graining(x,n)

% Function performs coarse graining
%
[B,A] = butter(6,1/n);       % Remove freq > fs/2n
y = filtfilt(B,A,x);         % Filter the data
y = y(1:n:end);              % resample by n
```

In the next example, we combine measurements of sample entropy with coarse graining to find the MSE of a random signal.

EXAMPLE 10.8

Calculate the MSE of Gaussian random noise used in Example 10.7. Use the same parameters as in Example 10.7 and calculate the sample entropy for 20 different scales. Plot sample entropy as a function of scale.

Solution: Modify the program in Example 10.7 so that the sample entropy calculation is placed in a loop where each iteration uses data at a different scale. The error tolerance, *r*, is set at 0.15 times the standard deviation of the original data after the mean is removed.[6] The first iteration should act on the unmodified data. Subsequent iterations should use `coarse graining` to alter the scale.

```
% Example 10.8 Evaluate MSE on Gaussian random data
%
N = 10000;                 % Data length
msf = 20;                  % Maximum scale factor
%
x = randn(1,N);            % Construct random data set
r = 0.15 * std(x);         % Determine error tolerance
for k = 1:msf
  if k == 1
    x_scaled = x;          % Original (unscaled) data
  else
    x_scaled = coarse_graining(x,k);     % Scale data
  end
  B = num_matches(x_scaled,2,r);         % Determine two-component matches
  A = num_matches(x_scaled,3,r);         % Determine three-component matches
  samp_entropy(k) = -log(A/B);           % Calculate sample entropy
  disp(['Scale factor: ',num2str(k)])    % Indicates progress
    end
    plot(samp_entropy);                  % Plot result
      .......labels.........
```

Result: Sample entropy as a function of scale is shown in Figure 10.15. An exponential decrease in entropy is seen, indicating that the data become less uncertain as the resolution is decreased. This is understandable because the original Gaussian random data are highly unpredictable and therefore have high entropy. Increasing the timescale reduces the noise and hence the uncertainty.

[6]Some researchers believe that the error tolerance, *r*, should be reset after each coarse-graining operation, as coarse-graining alters the standard deviation of the signal. Costa and colleagues, the originators of this technique, believe this is inappropriate, as the entropy calculation accounts for these changes. Here we use their original approach and do not reset the error tolerance, *r*, after coarse graining but acknowledge that this is controversial. Problem 18 explores the difference in results that occurs when the error tolerance is reset after coarse graining.

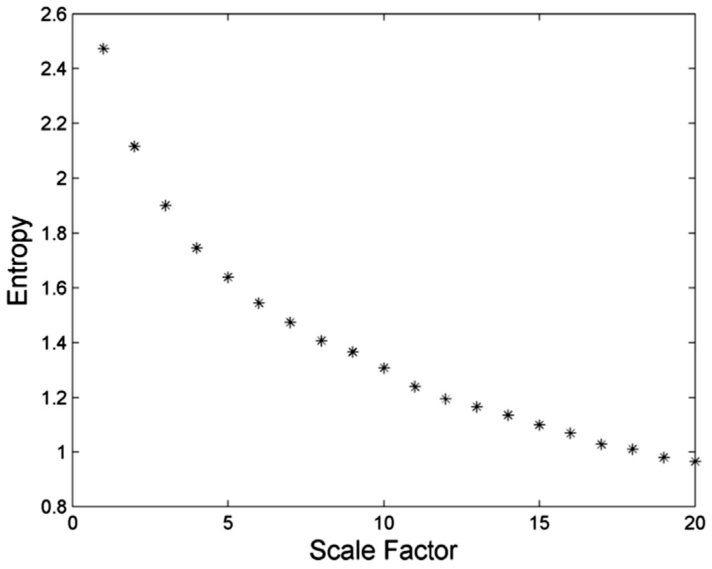

FIGURE 10.15 The change in sample entropy with scale. As the resolution decreases, the signal becomes more predictable, as is reflected in the decreased entropy.

Sample entropy and MSE have been used clinically to evaluate heart rate variability. These measures are often made differentially: the change in sample entropy of MSE evaluated at different time periods is used as a diagnostic indicator. Heart rate is influenced by numerous feedback pathways, so it tends to be less predictable in healthy subjects. Several different diseases, some cardiovascular in nature but others more general such as infection (i.e., sepsis), can reduce the effectiveness of these pathways, so the heart rate becomes more predictable and the entropy decreases. Entropy measurements have shown promise in assessing patient health for both cardiac and systemic disjunctions. One of the problems applies MSE to a signal representing heart rate variability signal.[7]

10.3.3 Detrended Fluctuation Analysis

DFA has been introduced by Peng (1994) to determine if a signal has fractal properties and, if so, to determine the fractal scaling. In a fractal signal, small segments of the signal are similar, in some sense, to larger segments. This similarity could be an actual matching of data but more likely some property of the data such as the variance. The amplitude of the property scales proportionally to the timescaling, but the constant of proportionality is not a constant, but rather a power function. This amplitude and timescaling relationship is stated mathematically as follows:

$$x(st) = s^H x(t) \tag{10.11}$$

[7]Heart rate data are usually recorded as a sequence of interbeat intervals, so-called RR intervals, and need to be converted to evenly spaced data using the approach presented in Example 4.4.

where x is the signal, or more generally, some property of the signal such as variance, s is the length of a signal segment, and H is a constant known as the Hurst coefficient. The proportionality constant, s^H, specifies the relationship of the property's value to the timescale. Thus a property of signal segment s is the same as that for the entire signal but diminished by s^H. DFA searches the signal for this relationship and attempts to determine H or rather a factor related to H. The property that is usually evaluated is the signal RMS or variance (essentially the same for long signals). DFA has been shown to be particularly effective at finding long-term correlations in the signal.

DFA has been widely applied to diverse fields, including DNA sequencing, heart rate dynamics, neuron spiking, human gait, and economic time series and also to weather-related and earthquake signals. In heart rate analysis, scaling behavior has been used to identify several different disease states.

To determine if various portions of a signal are statistically similar, we can measure the RMS (or variance, or other similar measurements) of small segments and compare them over time. This is the approach taken in Example 10.1 to determine if a signal is ergodic. If we find variations in our metric segment to segment, as we would expect in nonlinear data, they could be due to influences from nearby processes that have short timescales or influences from distant portions of the signal due to processes with long timescales. If the segments are small, we would expect that most of the influence would be from nearby, short-term processes because long-term processes have less of an influence on short segments; their primary effect would be on longer segments. This is illustrated in Figure 10.16 (upper panel) where two sinusoidal components, one at a lower frequency (dotted line) and other at a higher frequency (dashed lines), have been added to a random signal. If we look at a small segment of that signal (Figure 10.16 (lower left panel)) the higher frequency (i.e., short-term) signal has a significant impact on that signal (in terms of its RMS or variance). The low frequency signal has little effect, although both sinusoids have the same amplitude. We could reduce the influence of the high frequency signal on the short data segment by subtracting a polynomial fitted to the data samples in that signal. In fact, we could just subtract a straight line or linear component as was done in Figure 10.16 (lower right panel). As described earlier in Section 10.2, the linear component is called the "trend," so removing it is called "detrending." If we apply detrending to a longer signal segment, for example, the whole record shown in the Figure 10.16 (upper panel), we will reduce the influence of the long-term component (but not the short-term component). Thus, as we increase the segment size, detrending removes the influence of ever longer-term processes. This is the basic concept behind DFA.

The scaling relationship between amplitude and timescale (Equation 10.11) is only valid for nonstationary signals.[8] To ensure that DFA works on both stationary and nonstationary signals, the signal is integrated to generate what is known as the "signal profile." Integration is a linear process and does not change the fractal characteristics of the signal, but it does convert a stationary signal to a nonstationary signal. If the signal is nonstationary, integration makes no difference, it is still nonstationary. The signal profile is then divided into short segments, and the best straight-line fit for each segments is found based on least-square error (Figure 10.17). This straight line is then subtracted from the data, detrending the data

[8]For stationary signals the variance will be the same at all timescales. That was one of the tests that is described in Section 10.2 to see if a signal was stationary.

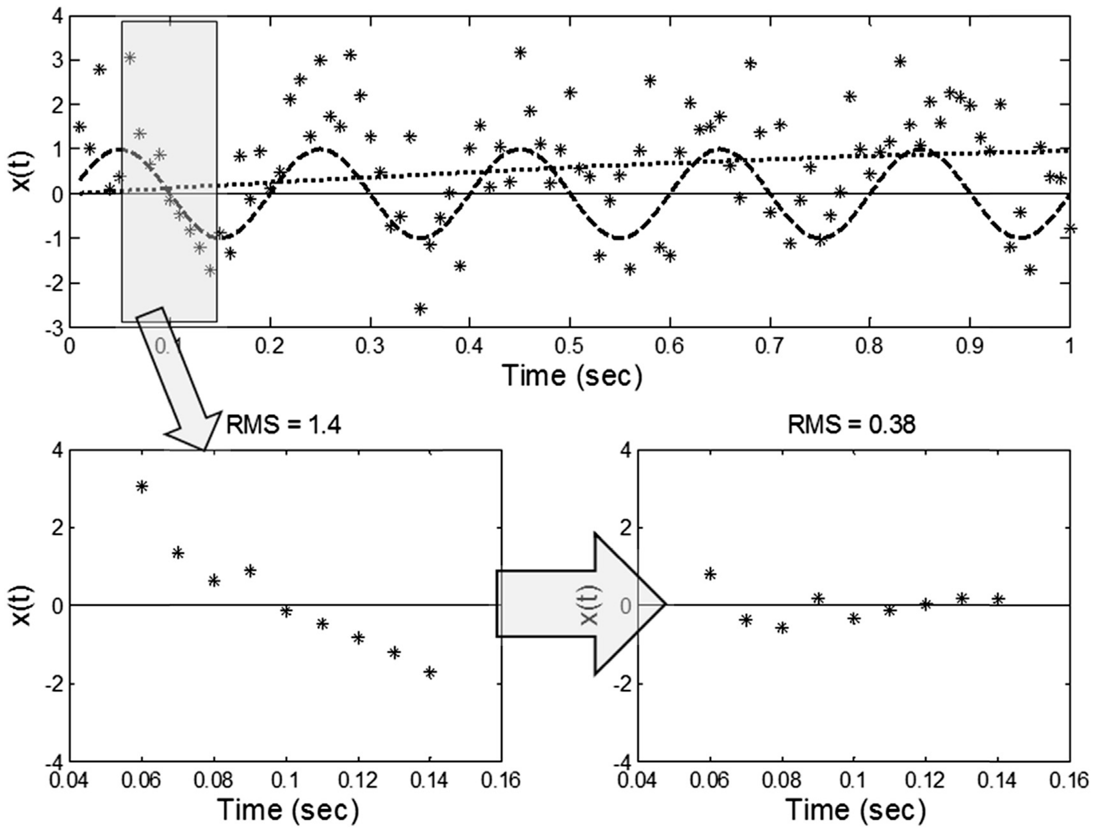

FIGURE 10.16 Illustration of the influence of short- and long-term processes on a short data segment. Upper panel: a random signal is influenced by both a short (dashed) and a long (dotted) sinusoidal process. Lower left panel: a small segment between 0.05 and 0.15 s is isolated from the overall signal and the influence of the short-term sinusoid is evident. Lower right panel: the influence of the short-term process has been reduced by subtracting the linear trend in the signal on the left. Removing the trend is called "detrending." Note the reduction in RMS after detrending.

segment. The RMS or variance of the detrended data segments is then taken. Rather than taking the average of RMS values from each section, it is simpler just to take the RMS of the overall detrended signal. To evaluate signal components having different timescales, the length of the detrended segments is varied. The RMS value represents the fluctuations of the detrended data, and they are plotted on a log–log scale against segment length.

Mathematically, the first step in the DFA, removal of the mean and digital integration, is represented as follows:

$$y_{int}[n] = \sum_{n=1}^{N} y[n] - \overline{y[n]} \tag{10.12}$$

where n is the time index and N is the number of samples in the signal.

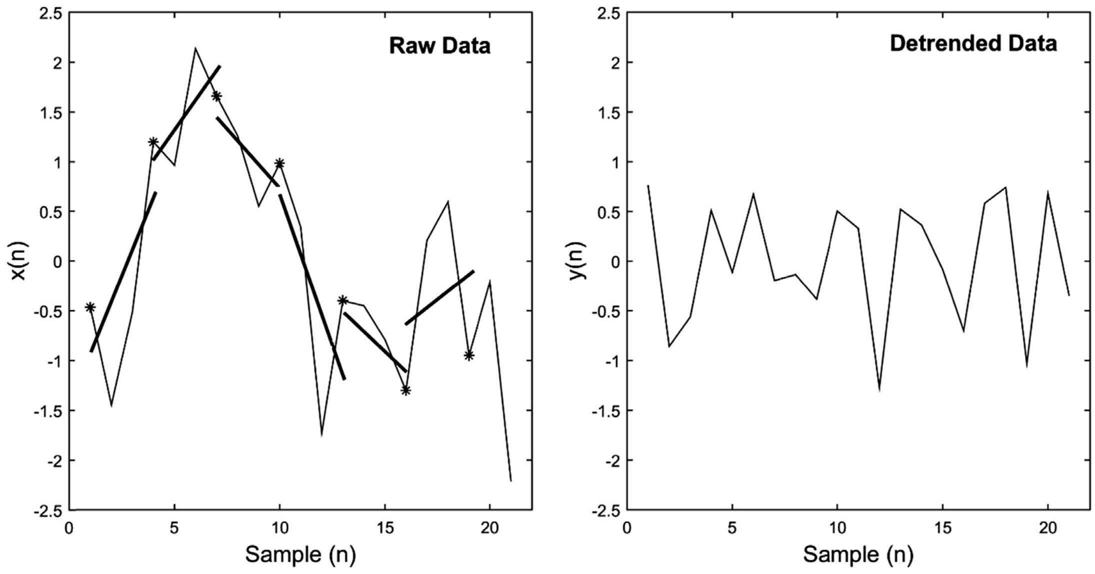

FIGURE 10.17 Detrending a signal. The raw data signal is divided into three-point segments. The * points indicated segment boundaries. Each segment is fitted by the best straight line based on least mean square error. These straight-line segments are subtracted from the raw data, leaving the detrended data shown in the right-hand plot. In DFA, the RMS of the detrended data is determined. These RMS values are determined over a range of segment lengths and plotted on a log–log scale against segment length.

The signal y_{int} is then divided into K segments of length m, and for each segment, a linear term, $y[n] - \overline{y[n]}$, is removed. (It is also possible to subtract out a higher-order polynomial.) After removing the linear trend from each segment, the remaining fluctuations are only those intrinsic signal variations minus the nearby (short-term) influences. The RMS value (or other property) is taken over the entire detrended signal length. These operations can be mathematically summarized as follows:

$$F(m) = \sqrt{\frac{1}{N}\sum_{k=1}^{K}[y_{\text{int}}(k) - y_{dt}(k)]^2} \tag{10.13}$$

where m is the segment length and $y_{int}(k)$ is the k^{th} segment from which a linear fit, y_{dt}, has been subtracted. This is performed using different segment lengths, m, and results in a different number of segments, K. As segment length increases, the short-term processes are included in the segment fluctuations, as long-term processes are removed. Because we are looking for an exponential scaling coefficient, the natural log of the RMS value, ln($F(m)$), is plotted against the natural log of m. The scaling coefficient related to H is found by searching for a straight-line section in the resultant curve.

A linear region on the log–log plot identifies a range where$F(m) \propto m^{\alpha}$, where α is a generalization of the Hurst coefficient, H. The exponent α indicates the nature of signal correlations where

$\alpha < 0.5$ indicates anti-correlated data,
$\alpha \approx 0.5$ indicates uncorrelated data,

$\alpha > 0.5$ indicates correlated data, and
$\alpha > 1$ indicates a non-stationary process or an unbounded correlation such as a sinusoid.

The problems examine how different signal features influence the value of α. An application of DFA is shown in the next example.

EXAMPLE 10.9

Find the DFA of a 10,000-sample Gaussian random signal.

Solution: Construct the signal using randn. Integrate the signal using MATLAB's `cumsum`. Detrending can be achieved using MATLAB's `detrend` routine as follows:

```
y = detrend(x);     % Detrend signal x
```

where `x` is the input signal and `y` is the detrended output signal.[9]

Because the results will be plotted on a log–log scale, we make the segment lengths in terms of powers of 2. The range should be determined by trial and error to search for a scaling region. In DFA it is common to use a minimum segment length of four samples (i.e., 2^2) and, given the data length of 10,000 samples, the maximum segment length should be 2^{13} (i.e, 8192) samples. The MATLAB code then is as follows:

```
% Example 10.9 Example of DFA

%
N = 10000;                  % Data length
X = rand(1,N);              1% Construct signal
m = 2.^(2:13);              % Segment lengths: 14 powers of 2
x = cumsum(randn(1,N));     % Integrated Gaussian random data
for n = 1:length(m);        % Loop for different segment lengths
  seg = m(n);               % Segment length
  K = floor(N/seg)-1;       % Number of segments to detrend
  for k = 1:K
    ix = seg*(k-1) +1;
    y(ix:ix+seg) = detrend(x(ix:ix+seg));  % Detrend
  end
  F(n) = sqrt(mean(y.^2));                 % Take RMS
end
m = log(m);
F = log(F);
plot(m,F,'b');
  .......labels and grid.......
% Use ginput to find the most linear section
[x,y] = ginput(2);                         % Get 2 points on the curve
plot(x,y,'k','LineWidth',2);               % Plot line superimposed
slope = (y(2) - y(1))/(x(2) - x(1))        % Calculate linear slope
```

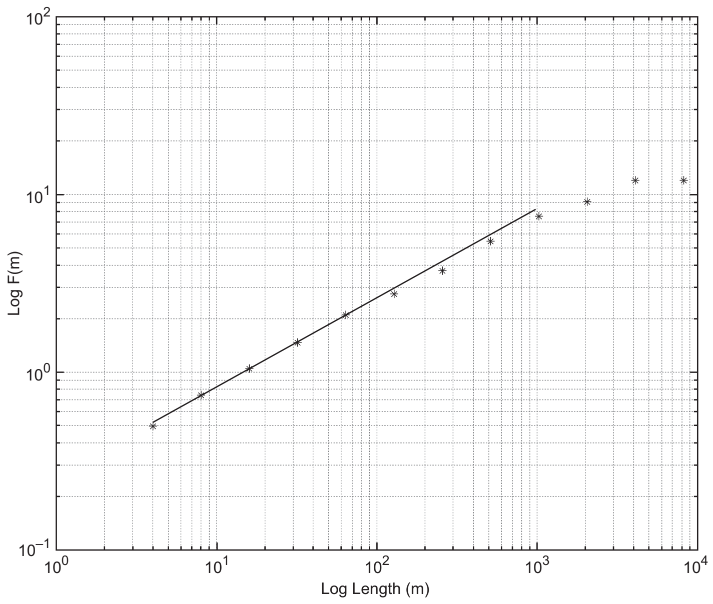

FIGURE 10.18 A plot of the log of detrended RMS signal value versus the log of segment length. The *solid reference line* has a slope of 0.5.

Result: The result is shown in Figure 10.18. Theoretically, Gaussian random data should have a scaling value, α, of 0.5 because it is uncorrelated. The plot of Figure 10.18 includes a reference line with a slope of 0.5. As it is seen here, the linear portion of the curve follows the reference line closely.

[9]There is a way to use detrending on segmented data with a single call, but it is much slower than using a loop to isolate the desired segments. In addition, it is not flexible, so it is not possible to use overlapping segments.

DFA is a technique that is relatively simple but powerful, if you believe that the data may contain signatures of long-term processes, but is dominated by unwanted or uninteresting events at the shorter terms.

10.4 SUMMARY

Stationarity is an important property of signals because all the signal analysis techniques we have studied thus far assume the signals are stationary. For a stationary signal, the basic signal properties of mean and variance do not change over time. For such a signal, the measurement of the mean or variance over only one segment is sufficient to estimate the signal's true mean. A more extreme case of stationarity is ergodicity, where not only are the mean and variance constant, but two higher moments, skew and kurtosis, are also constant over the length of signal. To determine whether a signal is stationary or ergodic requires multiple measurements over different signal segments and this usually entails a large amount of data.

Most biological signals have stationary means (termed ergodic in the mean) but are nonstationary in variance and other moments. We are, after all, creatures of change. In such cases, the most common remedy is to limit signal analyses to smaller segments where stationarity holds. Sometimes signal segmentation is arbitrary, but sometimes the underlying physiology suggests a segmentation strategy. With the ECG, a single heart beat is a natural segment, and an example of beat-to-beat segmentation and analysis of the ECG is presented in Example 10.3. This example introduces some basic concepts of pattern recognition, including feature detection, scattergrams, feature space boundaries, cluster analysis, and classification.

Signal nonlinearities can present analysis problems, as most methods we have studied thus far assume the signal is linear. However, nonlinearities also provide diagnostic opportunities. Current nonlinear analysis techniques attempt to identify fractal behavior and/or long-term correlations. Fractal behavior shows similarities within a signal when viewed at different timescales. This behavior is due to multiple physiological processes, each with a different timescale, acting on the signal. Analyses that seek to identify fractal behavior include correlation dimension, MSE, and DFA. The latter two are also useful for identifying long-term and short-term correlations.

Correlation dimension operates on a signal's phase space (or state space) trajectory to search for a fractal attractor, better known as a strange attractor. If only a one-dimensional signal is available, as is generally the case, a multidimensional phase trajectory can be approximated from delayed versions of the signal. The correlation sum is a measure of the space-filling characteristic of this phase trajectory. Fractal signals fill the space in which they are embedded and exhibit a fractal correlation dimension. Fractal correlation dimension is an indicator of fractal behavior. Unfortunately, the approach is sensitive to, and can be misled by, noise.

MSE grew out of information theory and attempts to estimate the entropy, or information content, of a signal at various resolutions. Information content is closely linked to signal unpredictability, so MSE looks for repeating patterns: the log ratio of two-sample patterns to three-point patterns is taken as a measure of unpredictability. Filtered downsampling is used to change signal scale, as each downsampling creates a signal at a lower scale (i.e., resolution).

DFA looks for fractal scaling and related long-term correlations. In fractal scaling, some signal property such as variance is found to be similar at different signal resolutions. However, the value or amplitude of this property is itself scaled in proportion to the timescaling, not linearly, but as an exponential. DFA measures this property over short signal segments, but first it removes any trend in the segment; that is, any linear change in signal baseline. Some versions even eliminate higher-order components in the segment. The idea is that such changes are due to nearby signal features, so if they are removed, the remaining signal contains only variation intrinsic to that segment. As the segments become longer, the detrending acts to reduce long-term interactions, but the larger segments now include the short-term fluctuations. Because we are looking for an exponential scale factor, the natural log of the RMS value is plotted against the natural log of the segment size. The slope of this curve is related to the exponential, H, known as the Hurst coefficient, and quantifies the scaling of the measured parameter. DFA is a simple technique, but not without controversy as some researchers suggest the approach has an inherent bias. Nonetheless, it is widely applied in biological signal analysis.

PROBLEMS

1. Repeat Example 10.2 using three different window sizes: 10, 60, and 240 s, all with 50% overlap. Plot the three results superimposed, and use a dashed line for the 10-s window and heavier linewidths for the longer data. Note how increasing window length smooths the data but with loss of detail.
2. Follow the approach of Example 10.2 and plot the alpha wave activity for two EEG signals in files `EEG_FP1_F4.mat` and `EEG_FP1_F3.mat`. Each signal is found as variable `val` in their respective files. These signals are taken at the same time from electrodes that were close together. Load both data files and analyze both signals simultaneously as in Example 10.2 using a 20-s window with a 50% overlap. Plot the resulting activities superimposed, but in different colors. Next take the cross-correlation between the two normalized RMS values. Is there a time delay between the two signals and if so what is its value? (Hint. You can use `crosscorr.m` to perform the cross-correlation, but remember that the correct lag value of each cross-correlation point is in the second output variable of this routine.)
3. Follow the approach of Example 10.2 to find the alpha wave and theta wave activity for the EEG signal given as variable `val` in the file `EEG_FP1_F3.mat`. The frequency range of the theta wave is 3.5 to 7.5 Hz. Plot the two RMS values superimposed. Use `corrcoef` to find the Pearson correlation between the two signals. (Hint. When using `corrcoef`, you may have to transpose the RMS data.)
4. Close examination of the clusters in Figure 10.3 suggests that the lower left cluster (cluster 1) may consist of a subcluster: a group of points that have a mean value less than 30. Modify the code of Example 10.4 to plot examples from two regions of cluster 1, above and below a mean of 30. Are the waveforms of these two subcultures different, i.e., do they represent two different classes? (Hint. When isolating the two subclusters, there are two criteria: mean $>$ or < -30 and RMS < 230. You must apply both or you could get samples from cluster 2 that are > -30.)
5. Modify Example 10.3 to change the second feature (i.e., `f2`). Divide each ECG segment in half and make the second feature the mean of the first half minus the mean of the second half. You should now get three different classes. Most beats cluster in the region where the first feature (RMS) is < 230 and the second feature is between 50 and 150. However, the beats having an RMS > 230 now fall into two different groups. Plot up to four beats from each of the two classes with RMS values > 230. Note the very different behaviors.
6. Apply cluster analysis to the EEG signal analyzed in Example 10.2 (variable `val` in the file `EEG_FP1_F7`). Use the same window size (10 s) and overlap (50%). For each isolated segment, extract two features: the normalized RMS of the alpha wave (as in Example 10.2) and the normalized RMS of the theta wave (3.5 to 7.5 Hz). You should be able to identify two possibly overlapping clusters. Use your best judgment on cluster boundaries and plot examples from each cluster. Because the signals are noisy, plot only three examples from each cluster and use different colors.
7. Modify Example 10.3 by adding an extra feature: the maximum value of the derivative. Use a derivative filter with a skip factor of 4. (Hint. If you have forgotten the

details of constructing a derivative filter (wonderfully presented in Section 8.5.1), you can borrow the code in `qrs_detect.m` given as part of Example 10.3.) Plot all three features on a 3D grid using `plot 3` (use option `grid on` to improve visibility of the clusters). Note that there is now a third small cluster (only three points) that has a higher peak derivative than the rest of cluster 2. (You may want to rotate the 3D plot for a better view of the three clusters.) Plot two examples from these two "high-RMS" clusters. (Hint. You will have to select waveforms that have high RMS (> 230) and higher or lower peak velocities. The boundary is around 600 to 800.) Note that this isolates the same ectopic beats found in Problem 5.

8. In Chapter 7, Example 7.8 uses the transfer function of human wrist mechanics to solve for the response of the wrist to an impulse of torque. The time domain solution was as follows:

$$\theta(t) = Ae^{-3t}(\sin(17.3t)\mathit{radians}$$

Plot the phase-plane trajectory of this response to three different values of A: $A = 0.1$, 0.173, and 0.25. Plot each trajectory in a different color and note that all trajectories end at the same fixed point attractor. Also plot the horizontal and vertical axes.

9. Use the simulation model constructed in Problem 16 of Chapter 9 to plot its phase-plane trajectory. This model is second order and based on the nonlinear van der Pol oscillator equation:

$$\frac{d^2x}{dt^2} = c\left(1 - x^2\right)\frac{dx}{dt} - x$$

Plot the phase plane of the system with an initial condition for x of 0.1 and a value for c of 2.0. This is an example of a limit cycle attractor. Simulate a 100-s time frame and plot the 0.0 horizontal and vertical axes. (Hint. The Simulink solver `ode23s` gives a smoother output than the default.)

10. Following Example 10.5, plot the 3D embedded trajectory of the EEG signal given as variable `x` in the file `EEG_1min.mat`. To find the appropriate delay, take the autocorrelation function, but you only need ±100 or so points on either side of zero lag. Use the lag at which the first zero occurs to set the delay and make $m = 3$. Does the resultant trajectory appear to fill the 3D space?

11. Apply correlation dimension analysis to a 10,000-sample Gaussian noise signal. To create an evenly spaced ln(r) for plotting, set r as an exponential, specifically use the MATLAB command: r = exp(−3:.05:−1). This gives about the same range of r as used in Example 10.6. Embed the Gaussian noise in 3D space using the routine `delay_emb`. Because Gaussian noise decorrelates after only one sample, you can use a short embedding delay, around four or five samples. One of the properties of Gaussian noise is that it tends to completely fill the phase space in which it is embedded, so you would expect a correlation dimension of around 3.0, the dimension of the phase space. (This problem may take about 9 min to run.)

12. Apply correlation dimension analysis to the signal heart rate signal given as variable `hr` in the file `hr_data.mat`. This is the same signal whose phase trajectory was developed in Example 10.5 and shown in Figure 10.9. Use the approach outlined in Example 10.6 and make r range between 0.02 and 0.14 in steps of 0.005. Use an embedding delay of 40 as justified in Example 10.5. Estimate the scaling dimension from a plot of $\ln C_r$ versus $\ln r$ as in Example 10.6. Make sure to eliminate the signal mean. (This problem may take about 12 min to run.)

13. Test the influence of noise in sample entropy. In a loop, use `sig_noise` (from Chapters 3 and 4) to construct a sine wave with added noise. To get a good estimate of sample entropy, make $N = 10{,}000$. Recall `sig_noise` assumes $f_s = 1000$ and make the sine wave 1 Hz (i.e., 10 cycles). Make sure the signal mean is 0 and scale by the standard deviation. Vary the signal-to-noise ratio (SNR) between −12 dB to +20 dB in 1 dB increments. Plot sample entropy against noise in dB. Note the dependence of sample entropy on SNR, although (as noted in the text) neither a sine wave nor the noise carries any information. (Run time: approximately 2.5 min.)

14. The file `rr_data.mat` contains two heart rate signals as a function of time. These data were obtained by applying `qrs_detect` to a 30 min ECG signal and taking the difference to find the RR interval. These RR intervals were converted to seconds by dividing by f_s (100 Hz). The approach described in Example 4.4 was used to convert the RR intervals to evenly spaced data ($f_{resamp} = 4$ Hz). The resulting signal was divided into two segments and stored as variables `rr1` and `rr2`.

Apply MSE to the two signals using a maximum scale factor of 20 and a maximum error of 0.15 times the standard deviation of the signal. Follow the procedure in Example 10.8, but run the code twice for each signal and superimpose the two plots using different colors or symbols. Both recordings show short-term correlations corresponding to a scale factor of 5 or 6. Although both show long-term interactions, one record has stronger long-term interactions. (Suggestion: You can do the two runs using a loop structure, which can be modified for use in the next four problems.)

15. Compute the MSE of two pure sine waves, one with a frequency of 0.5 Hz and the other with a frequency of 1 Hz. In theory the sine waves should contain no entropy and that is roughly the case for the unscaled entropy (i.e., scale factor = 1.0). The small increase in entropy observed is an artifact of the scaling process. As the signals are downsampled through scaling, the effective sampling frequency decreases and the sine waves become poorly sampled and less predictable, thus having higher entropy. (Note the entropy increase of the 1 Hz sine wave is greater than that of the 0.5 Hz sine wave.)

16. Compute the MSE of two sine waves used in Problem 15 but with a small amount of added Gaussian noise. Make the noise 0.1 times the amplitude of the sine waves. Compare your results with the MSE from pure Gaussian noise shown in Figure 10.15. Note that the unscaled entropy is much lower due to the low-entropy sine waves. As scaling increases, the entropy decreases as in Figure 10.15, but the sine wave artifact observed in Problem 15 eventually causes an increase in entropy. The scale at which the entropy begins to increase depends on the sine wave frequency as expected based on the results of Problem 15.

17. Generate a signal consisting of integrated Gaussian. (You can use MATLAB's `cumsum` to perform the integration.) This signal has long-term correlations and should show an increasing MSE with scaling. Apply MSE using three values of maximum error: 0.10, 0.15, and 0.20. Use a loop to compute the MSE for the three values of maximum error and plot the results superimposed using different colors or symbols. Note that as the maximum error increases, the MSE decreases, but that the overall shape of the curve is independent of maximum error.

18. Modify Example 10.8 so that the maximum error, r, is modified after each scaling to be 0.15 times the standard deviation of the scaled signal. The result shows entropy no longer decreases significantly with scale but stays roughly the same. (The increased variability seen at the higher scaling is probably due to the shorter data segments at these scales.) This indicates that the decrease in entropy with scaling is due largely to the reduction of amplitude produced by downsampling. As discussed in footnote 6, whether or not to readjust the maximum error after scaling remains controversial.

19. Apply DFA to the two sine waves used in Problem 15 and find α. Because the sine waves have unbounded correlations, we would expect α to be >1.0.

20. Apply DFA to the two heart rate signals used in Problem 14 (variables `rr1` and `rr2` in the `file rr_data.mat`). Find α for both from the most linear section of the curve. Because these signals likely contain long-term correlations we would expect α to be between 0.5 and 1.0. (Suggestion: It is difficult to identify the most linear section, but the slopes do not vary all that much, so any reasonable region should give a similar answer. Just be consistent for the two curves.)

21. Apply DFA to the EEG signal found as variable val in the file EEG_FP1_F3. Estimate α from the most linear portion of the curve. Because EEG signals are likely to be nonstationary, we would expect α to be >1.0. (Run time about 1 min.)

22. Analyze high pass–filtered and low pass–filtered noise using both DFA and MSE. The high pass–filtered noise is given as variable `xb(1,:)` in the file `prob10_22dat.mat` and the low pass–filtered noise is given as `xb(2,:)` in the same file. High pass–filtered signals are anticorrelated and low pass–filtered signals are correlated, so for the DFA analysis we would expect α to be <0.5 for the high pass–filtered noise and >0.5 for the low pass–filtered noise. For MSE, the low pass–filtered noise should show long-term correlations, but the anticorrelated (high pass–filtered) noise should show only short-term correlations. (Suggestion. Because the two analyses are very different, it is easier to use a separate program for each.)

23. Analyze one component of the Lorenz attractor and low pass–filtered (i.e., correlated) noise using both DFA and MSE. The x component of the Lorenz attractor system is given as variable `xa(1,:)` in the `file prob10_23dat.mat` and the filtered noise is given as `xa(2,:)` in the same file.

In the DFA analysis the Lorenz attractor shows a high α (> 1.0), whereas α for the filtered noise is near 0.5 because of the random noise. The MSE shows larger long-term correlations for the filtered noise, but not for the Lorenz system. However, because the Lorenz attractor is chaotic, it gives the same sample entropy across a wide range of scales. These results show that different nonlinear analyses provide different information about the signals. (Suggestion: Because the two results produce different graphs, it is easier to use a separate program for each analysis.)

24. Analyze low pass—filtered noise having two different bandwidths using both DFA and MSE. The filtered noise given as variable `xb(1,:)` in the file `prob22_24data.mat` has a bandwidth of 0.4 times $f_s/2$ whereas that in `xb(2,:)` of the same file has a bandwidth of 0.1 times $f_s/2$. Both signals were produced by filtering Gaussian noise with fourth-order Butterworth low-pass filters.

In the DFA analysis, the two signals produce similar curves with only slightly different slopes. In the MSE analysis, the signal with the larger bandwidth shows more short-term correlations that decrease with scale, whereas the lower bandwidth signal shows more long-term correlations. Again this shows that these two analyses provide different information.(Suggestion: Again because the two analyses are very different, it is easier to use a separate program for each.)

DETRENDED FLUCTUATION ANALYSIS

Ihlen, E. Introduction to multifractal detrended fluctuation analysis in Matlab. http://journal.frontiersin.org/article/10.3389/fphys.2012.00141/full.

Heartstone, R. et al. Detrended fluctuation analysis scale-free view on neuronal oscillations. Front. Physiol, November 30, 2012 https://doi.org/10.3389/fphys.2012.00450