Chapter 5

Exploratory Data Analysis

5.1 Introduction

Exploratory data analysis (EDA) is quantitative detective work according to John Tukey [1977]. EDA is the philosophy that data should first be explored without assumptions about probabilistic models, error distributions, number of groups, relationships between the variables, etc. for the purpose of discovering what they can tell us about the phenomena we are investigating. The goal of EDA is to explore the data to reveal patterns and features that will help the analyst better understand, analyze and model the data. With the advent of powerful desktop computers and high resolution graphics capabilities, these methods and techniques are within the reach of every statistician, engineer and data analyst.

EDA is a collection of techniques for revealing information about the data and methods for visualizing them to see what they can tell us about the underlying process that generated it. In most situations, exploratory data analysis should precede confirmatory analysis (e.g., hypothesis testing, ANOVA, etc.) to ensure that the analysis is appropriate for the data set. Some examples and goals of EDA are given below to help motivate the reader.

- If we have a time series, then we would plot the values over time to look for patterns such as trends, seasonal effects or change points. In Chapter 11, we have an example of a time series that shows evidence of a change point in a Poisson process.
- We have observations that relate two characteristics or variables, and we are interested in how they are related. Is there a linear or a nonlinear relationship? Are there patterns that can provide insight into the process that relates the variables? We will see examples of this application in Chapters 7 and 10.
- We need to provide some summary statistics that describe the data set. We should look for outliers or aberrant observations that might contaminate the results. If EDA indicates extreme observations are

in the data set, then robust statistical methods might be more appropriate. In Chapter 10, we illustrate an example where a graphical look at the data indicates the presence of outliers, so we use a robust method of nonparametric regression.

- We have a random sample that will be used to develop a model. This model will be included in our simulation of a process (e.g., simulating a physical process such as a queue). We can use EDA techniques to help us determine how the data might be distributed and what model might be appropriate.

In this chapter, we will be discussing graphical EDA and how these techniques can be used to gain information and insights about the data. Some experts include techniques such as smoothing, probability density estimation, clustering and principal component analysis in exploratory data analysis. We agree that these can be part of EDA, but we do not cover them in this chapter. Smoothing techniques are discussed in Chapter 10 where we present methods for nonparametric regression. Techniques for probability density estimation are presented in Chapter 8, but we do discuss simple histograms in this chapter. Methods for clustering are described in Chapter 9. Principal component analysis is not covered in this book, because the subject is discussed in many linear algebra texts [Strang, 1988; Jackson, 1991].

It is likely that some of the visualization methods in this chapter are familiar to statisticians, data analysts and engineers. As we stated in Chapter 1, one of the goals of this book is to promote the use of MATLAB for statistical analysis. Some readers might not be familiar with the extensive graphics capabilities of MATLAB, so we endeavor to describe the most useful ones for data analysis. In Section 5.2, we consider techniques for visualizing univariate data. These include such methods as stem-and-leaf plots, box plots, histograms, and quantile plots. We turn our attention to techniques for visualizing bivariate data in Section 5.3 and include a description of surface plots, scatterplots and bivariate histograms. Section 5.4 offers several methods for viewing multi-dimensional data, such as slices, isosurfaces, star plots, parallel coordinates, Andrews curves, projection pursuit, and the grand tour.

5.2 Exploring Univariate Data

Two important goals of EDA are: 1) to determine a reasonable model for the process that generated the data, and 2) to locate possible outliers in the sample. For example, we might be interested in finding out whether the distribution that generated the data is symmetric or skewed. We might also like to know whether it has one mode or many modes. The univariate visualization techniques presented here will help us answer questions such as these.

Histograms

A ***histogram*** is a way to graphically represent the frequency distribution of a data set. Histograms are a good way to

- summarize a data set to understand general characteristics of the distribution such as shape, spread or location,
- suggest possible probabilistic models, or
- determine unusual behavior.

In this chapter, we look only at the simple, basic histogram. Variants and extensions of the histogram are discussed in Chapter 8.

A ***frequency histogram*** is obtained by creating a set of bins or intervals that cover the range of the data set. It is important that these bins do not overlap and that they have equal width. We then count the number of observations that fall into each bin. To visualize this, we plot the frequency as the height of a bar, with the width of the bar representing the width of the bin. The histogram is determined by two parameters, the bin width and the starting point of the first bin. We discuss these issues in greater detail in Chapter 8. ***Relative frequency histograms*** are obtained by representing the height of the bin by the relative frequency of the observations that fall into the bin.

The basic MATLAB package has a function for calculating and plotting a univariate histogram. This function is illustrated in the example given below.

Example 5.1

In this example, we look at a histogram of the data in **forearm**. These data [Hand, et al., 1994; Pearson and Lee, 1903] consist of 140 measurements of the length in inches of the forearm of adult males. We can obtain a simple histogram in MATLAB using these commands:

```
load forearm
subplot(1,2,1)
% The hist function optionally returns the
% bin centers and frequencies.
[n,x] = hist(forearm);
% Plot and use the argument of width=1
% to produce bars that touch.
bar(x,n,1);
axis square
title('Frequency Histogram')
% Now create a relative frequency histogram.
% Divide each box by the total number of points.
subplot(1,2,2)
bar(x,n/140,1)
title('Relative Frequency Histogram')
axis square
```

These plots are shown in Figure 5.1. Notice that the shapes of the histograms are the same in both types of histograms, but the vertical axis is different. From the shape of the histograms, it seems reasonable to assume that the data are normally distributed.
❑

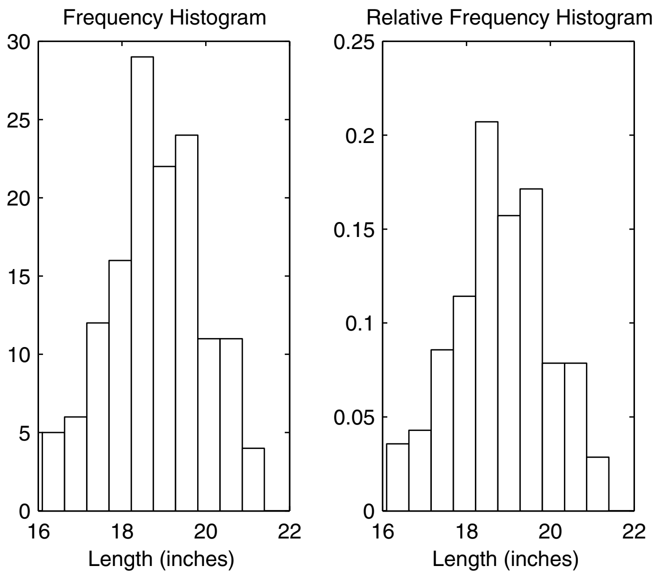

FIGURE 5.1
On the left is a frequency histogram of the `forearm` data, and on the right is the relative frequency histogram. These indicate that the distribution is unimodal and that the normal distribution is a reasonable model.

One problem with using a frequency or relative frequency histogram is that they do not represent meaningful probability densities, because they do not integrate to one. This can be seen by superimposing a corresponding normal distribution over the relative frequency histogram as shown in Figure 5.2.

A ***density histogram*** is a histogram that has been normalized so it will integrate to one. That means that if we add up the *areas* represented by the bars, then they should add up to one. A density histogram is given by the following equation

$$\hat{f}(x) = \frac{\nu_k}{nh} \qquad x \text{ in } B_k, \tag{5.1}$$

where B_k denotes the k-th bin, ν_k represents the number of data points that fall into the k-th bin and h represents the width of the bins. In the following

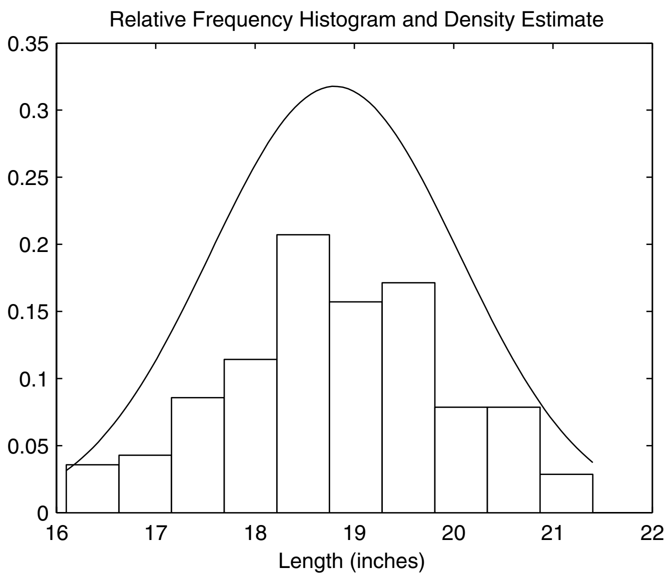

FIGURE 5.2
This shows a relative frequency histogram of the **forearm** data. Superimposed on the histogram is the normal probability density function using parameters estimated from the data. Note that the curve is higher than the histogram, indicating that the histogram is not a valid probability density function.

example, we reproduce the histogram of Figure 5.2 using the density histogram.

Example 5.2

Here we explore the **forearm** data using a density histogram. Assuming a normal distribution and estimating the parameters from the data, we can superimpose a smooth curve that represents an estimated density for the normal distribution.

```
% Get parameter estimates for the normal distribution.
mu = mean(forearm);
v = var(forearm);
% Obtain normal pdf based on parameter estimates.
xp = linspace(min(forearm),max(forearm));
yp = normp(xp,mu,v);
% Get the information needed for a histogram.
[nu,x] = hist(forearm);
% Get the widths of the bins.
h = x(2)-x(1);
```

```
% Plot as density histogram - Equation 5.1.
bar(x,nu/(140*h),1)
hold on
plot(xp,yp)
xlabel('Length (inches)')
title('Density Histogram and Density Estimate')
hold off
```

The results are shown in Figure 5.3. Note that the assumption of normality for the data is not unreasonable. The estimated density function and the density histogram match up quite well.
❑

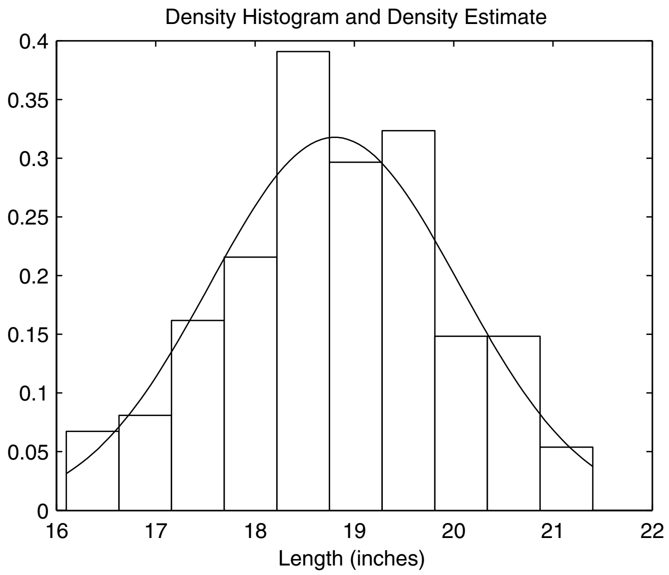

FIGURE 5.3
Density histogram for the **forearm** data. The curve represents a normal probability density function with parameters given by the sample mean and sample variance of the data. From this we see that the normal distribution is a reasonable probabilistic model.

Stem-and-Leaf

Stem-and-leaf plots were introduced by Tukey [1977] as a way of displaying data in a structured list. Presenting data in a table or an ordered list does not readily convey information about how the data are distributed, as is the case with histograms.

If we have data where each observation consists of at least two digits, then we can construct a stem-and-leaf diagram. To display these, we separate each measurement into two parts: the ***stem*** and the ***leaf***. The stems are comprised of the leading digit or digits, and the remaining digit makes up the leaf. For example, if we had the number 75, then the stem is the 7, and the leaf is the 5. If the number is 203, then the stem is 20 and the leaf is 3.

The stems are listed to the left of a vertical line with all of the leaves corresponding to that stem listed to the right. If the data contain decimal places, then they can be rounded for easier display. An alternative is to move the decimal place to specify the appropriate leaf unit. We provide a function with the text that will construct stem-and-leaf plots, and its use is illustrated in the next example.

Example 5.3

The heights of 32 Tibetan skulls [Hand, et al. 1994; Morant, 1923] measured in millimeters is given in the file **tibetan**. These data comprise two groups of skulls collected in Tibet. One group of 17 skulls comes from graves in Sikkim and nearby areas of Tibet and the other 15 skulls come from a battlefield in Lhasa. The original data contain five measurements, but for this example, we only use the fourth measurement. This is the upper face height, and we round to the nearest millimeter. We use the function **csstemleaf** that is provided with the text.

```
load tibetan
% This loads up all 5 measurements of the skulls.
% We use the fourth characteristic to illustrate
% the stem-and-leaf plot. We first round them.
x = round(tibetan(:,4));
csstemleaf(x)
title('Height (mm) of Tibetan Skulls')
```

The resulting stem-and-leaf is shown in Figure 5.4. From this plot, we see there is not much evidence that there are two groups of skulls, if we look only at the characteristic of upper face height. We will explore these data further in Chapter 9, where we apply pattern recognition methods to the problem.
❑

It is possible that we do not see much evidence for two groups of skulls because there are too few stems. EDA is an iterative process, where the analyst should try several visualization methods in search of patterns and information in the data. An alternative approach is to plot more than one line per stem. The function **csstemleaf** has an optional argument that allows the user to specify two lines per stem. The default value is one line per stem, as we saw in Example 5.3. When we plot two lines per stem, leaves that correspond to the digits 0 through 4 are plotted on the first line and those that have digits 5 through 9 are shown on the second line. A stem-and-leaf with two lines per stem for the Tibetan skull data is shown in Figure 5.5. In practice, Height (mm) of Tibetan Skulls

6	2 3 5 5 6 8 9
7	0 0 1 1 1 2 2 3 4 4 4 4 5 6 6 7 7 7 8 9 9
8	0 1 2 3

FIGURE 5.4
This shows the stem-and-leaf plot for the upper face height of 32 Tibetan skulls. The data have been rounded to the nearest millimeter.

Height (mm) of Tibetan Skulls

6	2 3
6	5 5 6 8 9
7	0 0 1 1 1 2 2 3 4 4 4 4
7	5 6 6 7 7 7 8 9 9
8	0 1 2 3
8	

FIGURE 5.5
This shows a stem-and-leaf plot for the upper face height of 32 Tibetan skulls where we now have two lines per stem. Note that we see approximately the same information (a unimodal distribution) as in Figure 5.4.

one could plot a stem-and-leaf with one and with two lines per stem as a way of discovering more about the data. The stem-and-leaf is useful in that it approximates the shape of the density, and it also provides a listing of the data. One can usually recover the original data set from the stem-and-leaf (if it has not been rounded), unlike the histogram. A disadvantage of the stem-and-leaf plot is that it is not useful for large data sets, while a histogram is very effective in reducing and displaying massive data sets.

Quantile-Based Plots - Continuous Distributions

If we need to compare two distributions, then we can use the quantile plot to visually compare them. This is also applicable when we want to compare a distribution and a sample or to compare two samples. In comparing the distributions or samples, we are interested in knowing how they are shifted relative to each other. In essence, we want to know if they are distributed in the same way. This is important when we are trying to determine the distribution that generated our data, possibly with the goal of using that information to generate data for Monte Carlo simulation. Another application where this is useful is in checking model assumptions, such as normality, before we conduct our analysis.

In this part, we discuss several versions of quantile-based plots. These include ***quantile-quantile plots*** (q-q plots) and ***quantile plots*** (sometimes called a ***probability plot***). Quantile plots for discrete data are discussed next. The quantile plot is used to compare a sample with a theoretical distribution. Typically, a q-q plot (sometimes called an ***empirical quantile plot***) is used to determine whether two random samples are generated by the same distribution. It should be noted that the q-q plot can also be used to compare a random sample with a theoretical distribution by generating a sample from the theoretical distribution as the second sample.

Q-Q Plot

The q-q plot was originally proposed by Wilk and Gnanadesikan [1968] to visually compare two distributions by graphing the quantiles of one versus the quantiles of the other. Say we have two data sets consisting of univariate measurements. We denote the order statistics for the first data set by

$$x_{(1)}, x_{(2)}, \dots, x_{(n)} .$$

Let the order statistics for the second data set be

$$y_{(1)}, y_{(2)}, \dots, y_{(m)} ,$$

with $m \leq n$.

We look first at the case where the sizes of the data sets are equal, so $m = n$. In this case, we plot as points the sample quantiles of one data set versus the other data set. This is illustrated in Example 5.4. If the data sets come from the same distribution, then we would expect the points to approximately follow a straight line.

A major strength of the quantile-based plots is that they do not require the two samples (or the sample and theoretical distribution) to have the same location and scale parameter. If the distributions are the same, but differ in location or scale, then we would still expect the quantile-based plot to produce a straight line.

Example 5.4

We will generate two sets of normal random variables and construct a q-q plot. As expected, the q-q plot (Figure 5.6) follows a straight line, indicating that the samples come from the same distribution.

```
% Generate the random variables.
x = randn(1,75);
y = randn(1,75);
% Find the order statistics.
xs = sort(x);
ys = sort(y);
% Now construct the q-q plot.
plot(xs,ys,'o')
xlabel('X - Standard Normal')
ylabel('Y - Standard Normal')
axis equal
```

If we repeat the above MATLAB commands using a data set generated from an exponential distribution and one that is generated from the standard normal, then we have the plot shown in Figure 5.7. Note that the points in this q-q plot do not follow a straight line, leading us to conclude that the data are not generated from the same distribution.
❑

We now look at the case where the sample sizes are not equal. Without loss of generality, we assume that $m < n$. To obtain the q-q plot, we graph the $y_{(i)}$, $i = 1, \ldots, m$ against the $(i - 0.5)/m$ quantile of the other data set. Note that this definition is not unique [Cleveland, 1993]. The $(i - 0.5)/m$ quantiles of the x data are usually obtained via interpolation, and we show in the next example how to use the function **csquantiles** to get the desired plot.

Users should be aware that q-q plots provide a rough idea of how similar the distribution is between two random samples. If the sample sizes are small, then a lot of variation is expected, so comparisons might be suspect. To help aid the visual comparison, some q-q plots include a reference line. These are lines that are estimated using the first and third quartiles $(q_{0.25}, q_{0.75})$ of each data set and extending the line to cover the range of the data. The

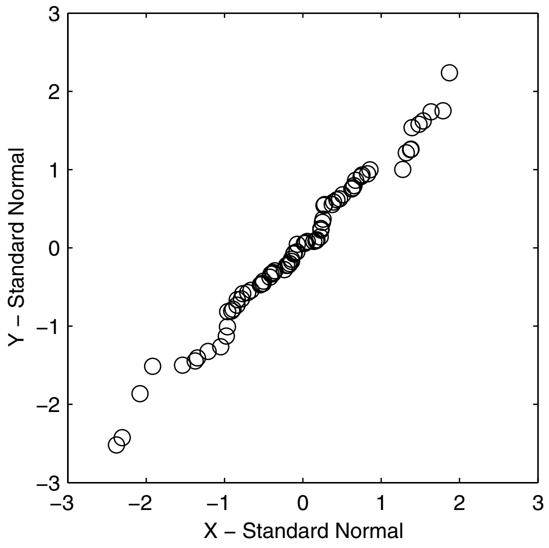

FIGURE 5.6
This is a q-q plot of x and y where both data sets are generated from a standard normal distribution. Note that the points follow a line, as expected.

MATLAB Statistics Toolbox provides a function called **qqplot** that displays this type of plot. We show below how to add the reference line.

Example 5.5

This example shows how to do a q-q plot when the samples do not have the same number of points. We use the function **csquantiles** to get the required sample quantiles from the data set that has the larger sample size. We then plot these versus the order statistics of the other sample, as we did in the previous examples. Note that we add a reference line based on the first and third quartiles of each data set, using the function **polyfit** (see Chapter 7 for more information on this function).

```
% Generate the random variables.
m = 50;
n = 75;
x = randn(1,n);
y = randn(1,m);
% Find the order statistics for y.
ys = sort(y);
% Now find the associated quantiles using the x.
% Probabilities for quantiles:
p = ((1:m) - 0.5)/m;
```

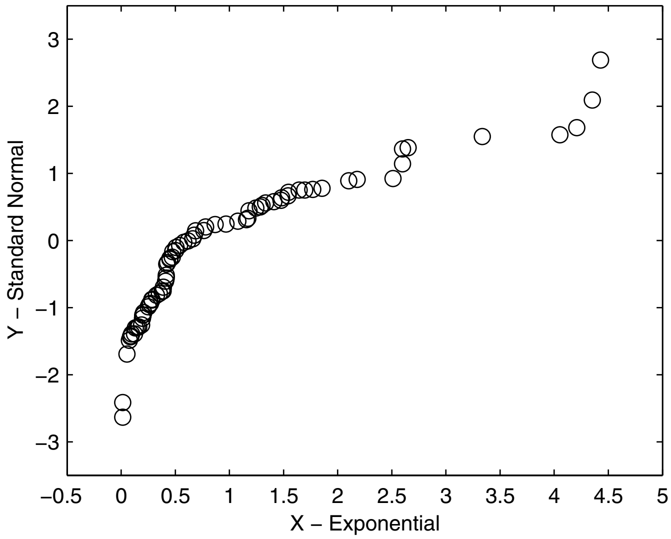

FIGURE 5.7
This is a q-q plot where one random sample is generated from the exponential distribution and one is generated by a standard normal distribution. Note that the points do not follow a straight line, indicating that the distributions that generated the random variables are not the same.

```
xs = csquantiles(x,p);
% Construct the plot.
plot(xs,ys,'ko')
% Get the reference line.
% Use the 1st and 3rd quartiles of each set to
% get a line.
qy = csquantiles(y,[0.25,0.75]);
qx = csquantiles(x,[0.25,0.75]);
[pol, s] = polyfit(qx,qy,1);
% Add the line to the figure.
yhat = polyval(pol,xs);
hold on
plot(xs,yhat,'k')
xlabel('Sample Quantiles - X'),
ylabel('Sorted Y Values')
hold off
```

From Figure 5.8, the assumption that each data set is generated according to the same distribution seems reasonable.
❑

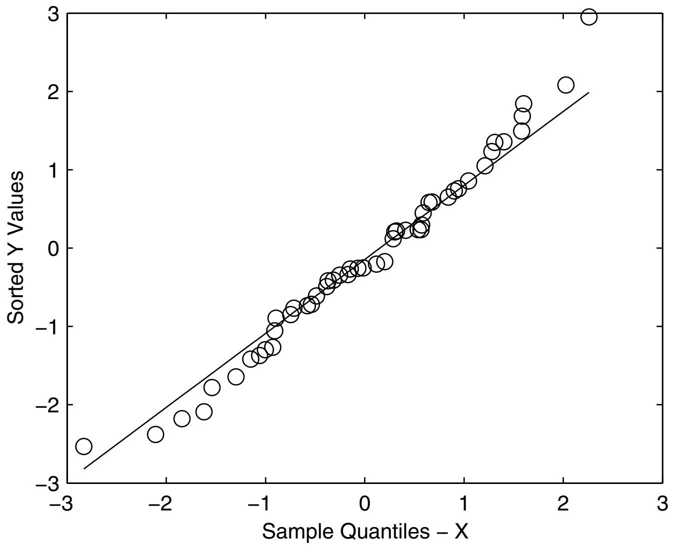

FIGURE 5.8
Here we show the q-q plot of Example 5.5. In this example, we also show the reference line estimated from the first and third quartiles. The q-q plot shows that the data do seem to come from the same distribution.

Quantile Plots

A ***quantile plot*** or ***probability plot*** is one where the theoretical quantiles are plotted against the order statistics for the sample. Thus, on one axis we plot the $x_{(i)}$ and on the other axis we plot

$$F^{-1}\left(\frac{i-0.5}{n}\right),$$

where $F^{-1}(.)$ denotes the inverse of the cumulative distribution function for the hypothesized distribution. As before, the 0.5 in the above argument can be different [Cleveland, 1993]. A well-known example of a quantile plot is the ***normal probability plot,*** where the ordered sample versus the quantiles of the normal distribution are plotted.

The MATLAB Statistics Toolbox has two functions for obtaining quantile plots. One is called **`normplot`**, and it produces a normal probability plot. So, if one would like to assess the assumption that a data set comes from a normal distribution, then this is the one to use. There is also a function for constructing a quantile plot that compares a data set to the Weibull distribution. This is called **`weibplot`**. For quantile plots with other theoretical distributions, one can use the MATLAB code given below, substituting the appropriate function to get the theoretical quantiles.

Example 5.6

This example illustrates how you can display a quantile plot in MATLAB. We first generate a random sample from the standard normal distribution as our data set. The sorted sample is an estimate of the $(i-0.5)/n$ quantile, so we next calculate these probabilities and get the corresponding theoretical quantiles. Finally, we use the function **norminv** from the Statistics Toolbox to get the theoretical quantiles for the normal distribution. The resulting quantile plot is shown in Figure 5.9.

```
% Generate a random sample from a standard normal.
x = randn(1,100);
% Get the probabilities.
prob = ((1:100)-0.5)/100;
% Now get the theoretical quantiles.
qp = norminv(prob,0,1);
% Now plot theoretical quantiles versus
% the sorted data.
plot(sort(x),qp,'ko')
xlabel('Sorted Data')
ylabel('Standard Normal Quantiles')
```

To further illustrate these concepts, let's see what happens when we generate a random sample from a uniform $(0, 1)$ distribution and check it against the normal distribution. The MATLAB code is given below, and the quantile plot is shown in Figure 5.10. As expected, the points do not lie on a line, and we see that the data are not from a normal distribution.

```
% Generate a random sample from a
% uniform distribution.
x = rand(1,100);
% Get the probabilities.
prob = ((1:100)-0.5)/100;
% Now get the theoretical quantiles.
qp = norminv(prob,0,1);
% Now plot theoretical quantiles versus
% the sorted data.
plot(sort(x),qp,'ko')
ylabel('Standard Normal Quantiles')
xlabel('Sorted Data')
```

❑

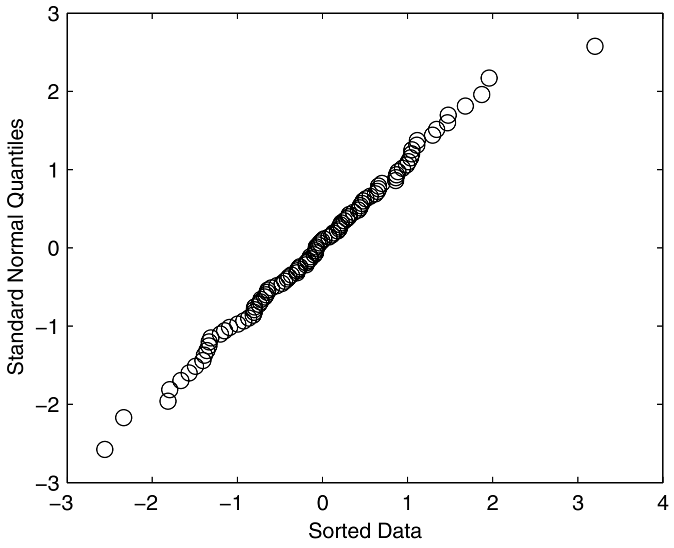

FIGURE 5.9
This is a quantile plot or normal probability plot of a random sample generated from a standard normal distribution. Note that the points approximately follow a straight line, indicating that the normal distribution is a reasonable model for the sample.

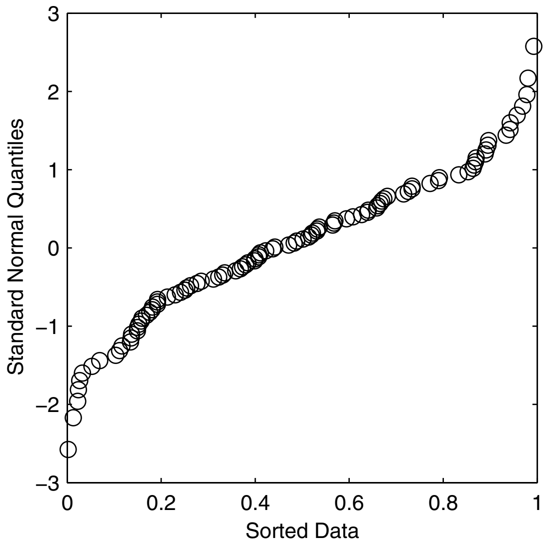

FIGURE 5.10
Here we have a quantile plot where the sample is generated from a uniform distribution, and the theoretical quantiles are from the normal distribution. The shape of the curve verifies that the sample is not from a normal distribution.

Quantile Plots - Discrete Distributions

Previously, we discussed quantile plots that are primarily used for continuous data. We would like to have a similar technique for graphically comparing the shapes of discrete distributions. Hoaglin and Tukey [1985] developed several plots to accomplish this. We present two of them here: the ***Poissonness plot*** and the ***binomialness plot***. These will enable us to search for evidence that our discrete data follow a Poisson or a binomial distribution. They also serve to highlight which points might be incompatible with the model.

Poissonness Plot

Typically, discrete data are whole number values that are often obtained by counting the number of times something occurs. For example, these might be the number of traffic fatalities, the number of school-age children in a household, the number of defects on a hard drive, or the number of errors in a computer program. We sometimes have the data in the form of a frequency distribution that lists the possible count values (e.g., 0.1, 2, ...) and the number of observations that are equal to the count values.

The counts will be denoted as k, with $k = 0, 1, \ldots, L$. We will assume that L is the maximum observed value for our discrete variable or counts in the data set and that we are interested in all counts between 0 and L. Thus, the total number of observations in the sample is

$$N = \sum_{k=0}^{L} n_k ,$$

where n_k represents the number of observations that are equal to the count k.

A basic Poissonness plot is constructed by plotting the count values k on the horizontal axis and

$$\varphi(n_k) = \ln(k!n_k/N) \qquad (5.2)$$

on the vertical axis. These are plotted as symbols, similar to the quantile plot. If a Poisson distribution is a reasonable model for the data, then this should follow a straight line. Systematic curvature in the plot would indicate that these data are not consistent with a Poisson distribution. The values for $\varphi(n_k)$ tend to have more variability when n_k is small, so Hoaglin and Tukey [1985] suggest plotting a special symbol or a '1' to highlight these points.

Example 5.7

This example is taken from Hoaglin and Tukey [1985]. In the late 1700's, Alexander Hamilton, John Jay and James Madison wrote a series of 77 essays under the title of *The Federalist*. These appeared in the newspapers under a

TABLE 5.1

Frequency distribution of the word *may* in essays known to be written by James Madison. The n_k represent the number of blocks of text that contained k occurrences of the word *may* [Hoaglin and Tukey, 1985].

Number of Occurrences of the Word *may* (k)	Number of Blocks (n_k)
0	156
1	63
2	29
3	8
4	4
5	1
6	1

pseudonym. Most analysts accept that John Jay wrote 5 essays, Alexander Hamilton wrote 43, Madison wrote 14, and 3 were jointly written by Hamilton and Madison. Later, Hamilton and Madison claimed that they each solely wrote the remaining 12 papers. To verify this claim, Mosteller and Wallace [1964] used statistical methods, some of which were based on the frequency of words in blocks of text. Table 5.1 gives the frequency distribution for the word *may* in papers that were known to be written by Madison. We are not going to repeat the analysis of Mosteller andWallace, we are simply using the data to illustrate a Poissonness plot. The following MATLAB code produces the Poissonness plot shown in Figure 5.11.

```
k = 0:6;  % vector of counts
n_k = [156 63 29 8 4 1 1];
N=sum(n_k);
% Get vector of factorials.
fact = zeros(size(k));
for i = k
   fact(i+1) = factorial(i);
end
% Get phi(n_k) for plotting.
phik = log(fact.*n_k/N);
% Find the counts that are equal to 1.
% Plot these with the symbol 1.
% Plot rest with a symbol.
ind = find(n_k~=1);
plot(k(ind),phik(ind),'o')
ind = find(n_k==1);
if ~isempty(ind)
   text(k(ind),phik(ind),'1')
```

```
end
% Add some whitespace to see better.
axis([-0.5 max(k)+1 min(phik)-1 max(phik)+1])
xlabel('Number of Occurrences - k')
ylabel('\phi (n_k)')
```

The Poissonness plot has significant curvature indicating that the Poisson distribution is not a good model for these data. There are also a couple of points with a frequency of 1 that seem incompatible with the rest of the data. Thus, if a statistical analysis of these data relies on the Poisson model, then any results are suspect.

❑

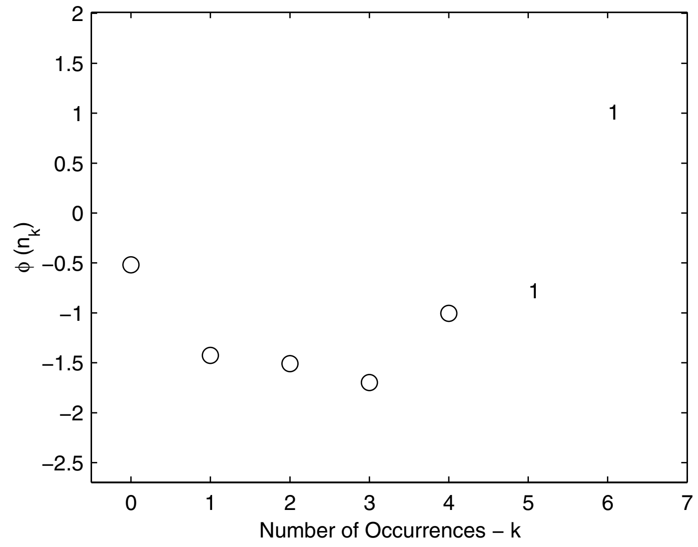

FIGURE 5.11
This is a basic Poissonness plot using the data in Table 5.1. The symbol *1* indicates that $n_k = 1$.

Hoaglin and Tukey [1985] suggest a modified Poissonness plot that is obtained by changing the n_k, which helps account for the variability of the individual values. They propose the following change:

$$n_k^* = \begin{cases} n_k - 0.67 - 0.8n_k/N; & n_k \geq 2 \\ 1/e; & n_k = 1 \\ \text{undefined}; & n_k = 0. \end{cases} \tag{5.3}$$

As we will see in the following example where we apply the modified Poissonness plot to the word frequency data, the main effect of the modified plot is to highlight those data points with small counts that do not behave contrary to the other observations. Thus, if a point that is plotted as a *1* in a modified Poissonness plot seems different from the rest of the data, then it should be investigated.

Example 5.8

We return to the word frequency data in Table 5.1 and show how to get a modified Poissonness plot. In this modified version shown in Figure 5.12, we see that the points where $n_k = 1$ do not seem so different from the rest of the data.

```
% Poissonness plot - modified
k = 0:6;  % vector of counts
% Find n*_k.
n_k = [156 63 29 8 4 1 1];
N = sum(n_k);
phat = n_k/N;
nkstar = n_k-0.67-0.8*phat;
% Get vector of factorials.
fact = zeros(size(k));
for i = k
   fact(i+1) = factorial(i);
end
% Find the frequencies that are 1; nkstar=1/e.
ind1 = find(n_k==1);
nkstar(ind1)= 1/2.718;
% Get phi(n_k) for plotting.
phik = log(fact.*nkstar/N);
ind = find(n_k~=1);
plot(k(ind),phik(ind),'o')
if ~isempty(ind1)
   text(k(ind1),phik(ind1),'1')
end
% Add some whitespace to see better.
axis([-0.5 max(k)+1 min(phik)-1 max(phik)+1])
xlabel('Number of Occurrences - k')
ylabel('\phi (n^*_k)')
```

❑

Binomialness Plot

A binomialness plot is obtained by plotting k along the horizontal axis and plotting

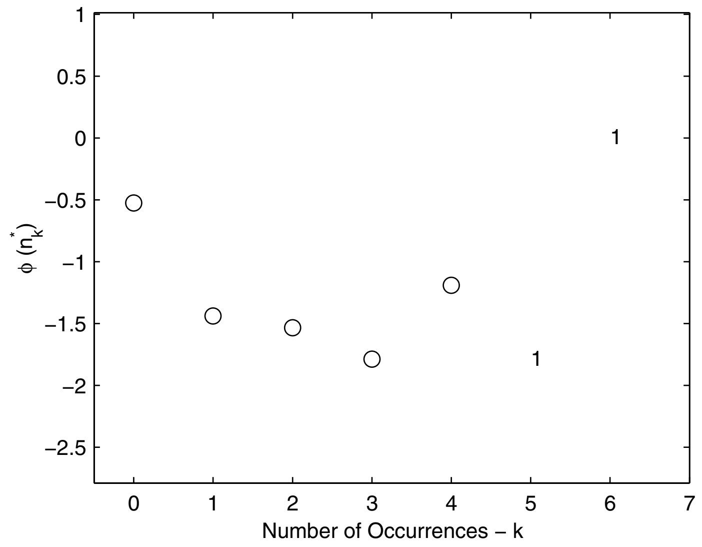

FIGURE 5.12
This is a modified Poissonness plot for the word frequency data in Table 5.1. Here the counts where $n_k = 1$ do not seem radically different from the rest of the observations.

$$\varphi(n_k^*) = \ln\left\{\frac{n_k^*}{N \times \binom{n}{k}}\right\}, \tag{5.4}$$

along the vertical axis. Recall that n represents the number of trials, and n_k^* is given by Equation 5.3. As with the Poissonness plot, we are looking for an approximate linear relationship between k and $\varphi(n_k^*)$. An example of the binomialness plot is given in Example 5.9.

Example 5.9

Hoaglin and Tukey [1985] provide a frequency distribution representing the number of females in 100 queues of length 10. These data are given in Table 5.2. The MATLAB code to display a binomialness plot for $n = 10$ is given below. Note that we cannot display $\varphi(n_k^*)$ for $k = 10$ (in this example), because it is not defined for $n_k = 0$. The resulting binomialness plot is shown in Figure 5.13, and it indicates a linear relationship. Thus, the binomial model for these data seems adequate.

```
   % Binomialness plot.
```

TABLE 5.2

Frequency Distribution for the Number of Females in a Queue of Size 10 [Hoaglin and Tukey, 1985]

Number of Females (k)	Number of Blocks (n_k)
0	1
1	3
2	4
3	23
4	25
5	19
6	18
7	5
8	1
9	1
10	0

```
k = 0:9;
n = 10;
n_k = [1 3 4 23 25 19 18 5 1 1];
N = sum(n_k);
nCk = zeros(size(k));
for i = k
   nCk(i+1) = cscomb(n,i);
end
phat = n_k/N;
nkstar = n_k-0.67-0.8*phat;
% Find the frequencies that are 1; nkstar=1/e.
ind1 = find(n_k==1);
nkstar(ind1) = 1/2.718;
% Get phi(n_k) for plotting.
phik = log(nkstar./(N*nCk));
% Find the counts that are equal to 1.
ind = find(n_k~=1);
plot(k(ind),phik(ind),'o')
if ~isempty(ind1)
   text(k(ind1),phik(ind1),'1')
end
% Add some whitespace to see better.
axis([-0.5 max(k)+1 min(phik)-1 max(phik)+1])
xlabel('Number of Females - k')
ylabel('\phi (n^*_k)')
```

❑

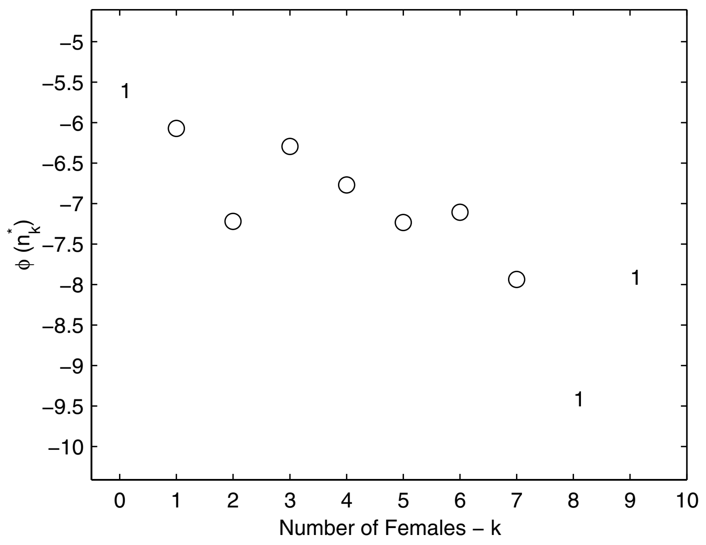

FIGURE 5.13
This shows the binomialness plot for the data in Table 5.2. From this it seems reasonable to use the binomial distribution to model the data.

Box Plots

Box plots (sometimes called box-and-whisker diagrams) have been in use for many years [Tukey, 1977]. As with most visualization techniques, they are used to display the distribution of a sample. Five values from a data set are used to construct the box plot. These are the three sample quartiles $(\hat{q}_{0.25}, \hat{q}_{0.5}, \hat{q}_{0.75})$, the minimum value in the sample and the maximum value.

There are many variations of the box plot, and it is important to note that they are defined differently depending on the software package that is used. Frigge, Hoaglin and Iglewicz [1989] describe a study on how box plots are implemented in some popular statistics programs such as Minitab, S, SAS, SPSS and others. The main difference lies in how outliers and quartiles are defined. Therefore, depending on how the software calculates these, different plots might be obtained [Frigge, Hoaglin and Iglewicz, 1989].

Before we describe the box plot, we need to define some terms. Recall from Chapter 3, that the ***interquartile range*** (IQR) is the difference between the first and the third sample quartiles. This gives the range of the middle 50% of the data. It is estimated from the following

$$I\hat{Q}R = \hat{q}_{0.75} - \hat{q}_{0.25} . \qquad (5.5)$$

Two limits are also defined: a lower limit (LL) and an upper limit (UL). These are calculated from the estimated IQR as follows

$$\begin{aligned} LL &= \hat{q}_{0.25} - 1.5 \cdot \hat{IQR} \\ UL &= \hat{q}_{0.75} + 1.5 \cdot \hat{IQR}. \end{aligned} \tag{5.6}$$

The idea is that observations that lie outside these limits are possible outliers. ***Outliers*** are data points that lie away from the rest of the data. This might mean that the data were incorrectly measured or recorded. On the other hand, it could mean that they represent extreme points that arise naturally according to the distribution. In any event, they are sample points that are suitable for further investigation.

Adjacent values are the most extreme observations in the data set that are within the lower and the upper limits. If there are no potential outliers, then the adjacent values are simply the maximum and the minimum data points.

To construct a box plot, we place horizontal lines at each of the three quartiles and draw vertical lines to create a box. We then extend a line from the first quartile to the smallest adjacent value and do the same for the third quartile and largest adjacent value. These lines are sometimes called the whiskers. Finally, any possible outliers are shown as an asterisk or some other plotting symbol. An example of a box plot is shown in Figure 5.14.

Box plots for different samples can be plotted together for visually comparing the corresponding distributions. The MATLAB Statistics Toolbox contains a function called **`boxplot`** for creating this type of display. It displays one box plot for each column of data. When we want to compare data sets, it is better to display a box plot with notches. These notches represent the uncertainty in the locations of central tendency and provide a rough measure of the significance of the differences between the values. If the notches do not overlap, then there is evidence that the medians are significantly different. The length of the whisker is easily adjusted using optional input arguments to **`boxplot`**. For more information on this function and to find out what other options are available, type **`help boxplot`** at the MATLAB command line.

Example 5.10

In this example, we first generate random variables from a uniform distribution on the interval $(0, 1)$, a standard normal distribution, and an exponential distribution. We will then display the box plots corresponding to each sample using the MATLAB function **`boxplot`**.

```
% Generate a sample from the uniform distribution.
xunif = rand(100,1);
% Generate sample from the standard normal.
xnorm = randn(100,1);
% Generate a sample from the exponential distribution.
```

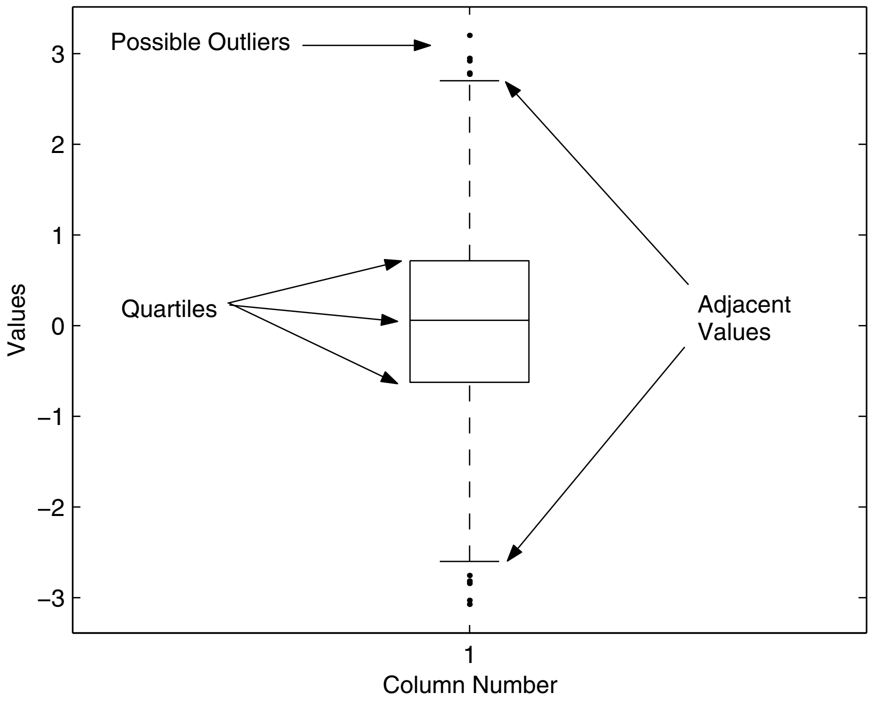

FIGURE 5.14
An example of a box plot with possible outliers shown as points.

```
% NOTE: this function is from the Statistics Toolbox.
xexp = exprnd(1,100,1);
boxplot([xunif,xnorm,xexp],1)
```

It can be seen in Figure 5.15 that the box plot readily conveys the shape of the distribution. A symmetric distribution will have whiskers with approximately equal lengths, and the two sides of the box will also be approximately equal. This would be the case for the uniform or normal distribution. A skewed distribution will have one side of the box and whisker longer than the other. This is seen in Figure 5.15 for the exponential distribution. If the interquartile range is small, then the data in the middle are packed around the median. Conversely, if it is large, then the middle 50% of the data are widely dispersed.
❑

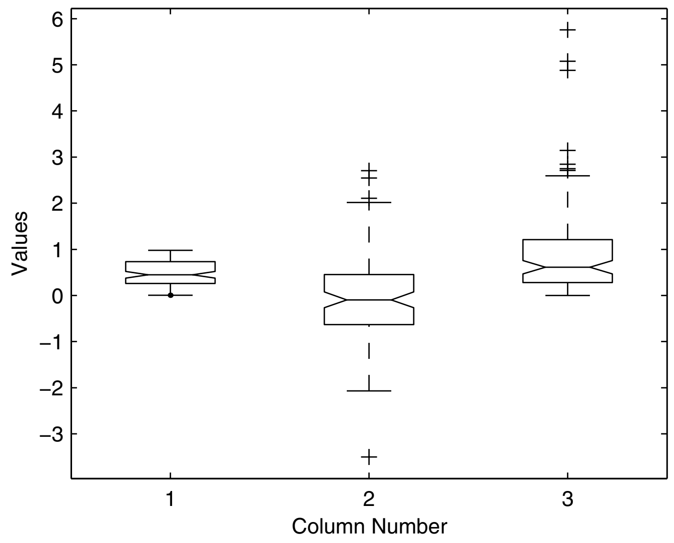

FIGURE 5.15
Here we have three box plots. The one on the left is for a sample from the uniform distribution. The data for the middle box plot came from a standard normal distribution, while the data for the box plot on the right came from an exponential. Notice that the shape of each distribution is apparent from the information contained in the box plots.

5.3 Exploring Bivariate and Trivariate Data

Using Cartesian coordinates, we can view up to three dimensions. For example, we could view bivariate data as points or trivariate data as a point cloud. We could also view a bivariate function, $z = f(x, y)$ as a surface. Visualizing anything more than three dimensions is very difficult, but we do offer some techniques in the next section. In this section, we present several methods for visualizing 2-D and 3-D data, looking first at bivariate data. Most of the techniques that we discuss are readily available in the basic MATLAB program.

Scatterplots

Perhaps one of the easiest ways to visualize bivariate data is with the scatterplot. A scatterplot is obtained by displaying the ordered pairs as points using some plotting symbol. This type of plot conveys useful information such as how the data are distributed in the two dimensions and how the two variables are related (e.g., a linear or a nonlinear relationship). Before any modeling, such as regression, is done using bivariate data, the analyst should always look at a scatterplot to see what type of relationship is reasonable. We will explore this further in Chapters 7 and 10.

A scatterplot can be obtained easily in MATLAB using the **plot** command. One simply enters the marker style or plotting symbol as one of the arguments. See the **help** on **plot** for more information on what characters are available. By entering a marker (or line) style, you tell MATLAB that you do *not* want to connect the points with a straight line, which is the default. We have already seen many examples of how to use the **plot** function in this way when we constructed the quantile and q-q plots.

An alternative function for scatterplots that is available with MATLAB is the function called **scatter**. This function takes the input vectors **x** and **y** and plots them as symbols. There are optional arguments that will plot the markers as different colors and sizes. These alternatives are explored in Example 5.11.

Example 5.11

We first generate a set of bivariate normal random variables using the technique described in Chapter 4. However, it should be noted that we find the matrix **R** in Equation 4.19 using singular value decomposition rather than Cholesky factorization. We then create a scatterplot using the **plot** function and the **scatter** function. The resulting plots are shown in Figure 5.16 and Figure 5.17.

```
% Create a positive definite covariance matrix.
vmat = [2, 1.5; 1.5, 9];
% Create mean at (2,3).
mu = [2 3];
[u,s,v] = svd(vmat);
vsqrt = ( v*(u'.*sqrt(s)))';
% Get standard normal random variables.
td = randn(250,2);
% Use x=z*sigma+mu to transform - see Chapter 4.
data = td*vsqrt+ones(250,1)*mu;
% Create a scatterplot using the plot function.
% Figure 5.16.
plot(data(:,1),data(:,2),'x')
axis equal
% Create a scatterplot using the scatter fumction.
% Figure 5.17.
% Use filled-in markers.
scatter(data(:,1),data(:,2),'filled')
axis equal
box on
```

❑

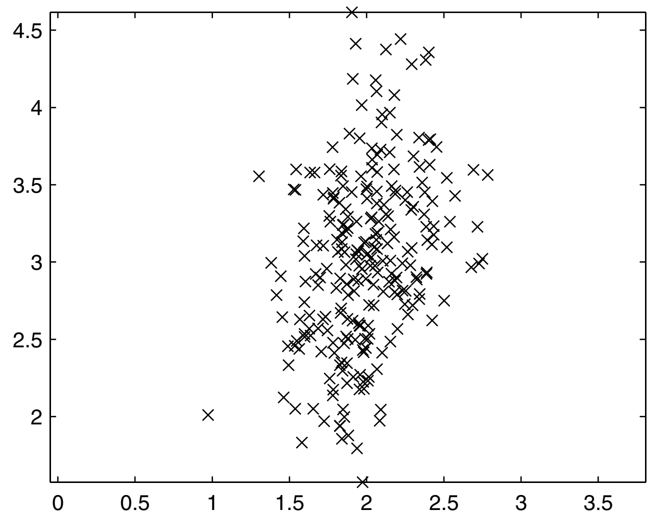

FIGURE 5.16
This is a scatterplot of the sample in Example 5.11 using the **plot** function. We can see that the data seem to come from a bivariate normal distribution. Here we use **'x'** as an argument to the **plot** function to plot the symbols as x's.

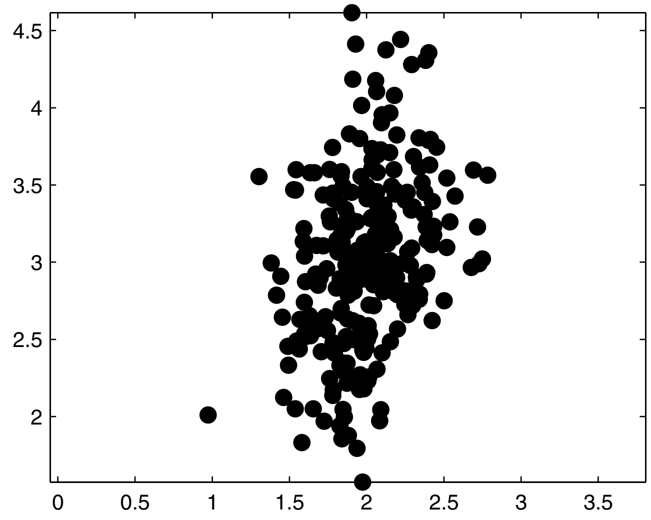

FIGURE 5.17
This is a scatterplot of the sample in Example 5.11 using the **scatter** function with filled markers.

Surface Plots

If we have data that represents a function defined over a bivariate domain, such as $z = f(x, y)$, then we can view our values for z as a surface. MATLAB provides two functions that display a matrix of z values as a surface: **mesh** and **surf**.

The **mesh** function displays the values as points above a rectangular grid in the x-y plane and connects adjacent points with straight lines. The mesh lines can be colored using various options, but the default method maps the height of the surface to a color.

The **surf** function is similar to **mesh**, except that the open spaces between the lines are filled in with color, with lines shown in black. Other options available with the **shading** command remove the lines or interpolate the color across the patches. An example of where the ability to display a surface can be used is in visualizing a probability density function (see Chapter 8).

Example 5.12

In this example, we begin by generating a grid over which we evaluate a bivariate normal density function. We then calculate the z values that correspond to the function evaluated at each x and y. We can display this as a surface using **surf**, which is shown in Figure 5.18.

```
% Create a bivariate standard normal.
% First create a grid for the domain.
[x,y] = meshgrid(-3:.1:3,-3:.1:3);
% Evaluate using the bivariate standard normal.
z = (1/(2*pi))*exp(-0.5*(x.^2+y.^2));
% Do the plot as a surface.
surf(x,y,z)
```

❑

Special effects can be achieved by changing color maps and using lighting. For example, lighting and color can help highlight structure or features on functions that have many bumps or a jagged surface. We will see some examples of how to use these techniques in the next section and in the exercises at the end of the chapter.

Contour Plots

We can also use contour plots to view our surface. Contour plots show lines of constant surface values, similar to topographical maps. Two functions are available in MATLAB for creating 2-D and 3-D contour plots. These are called **contour** and **contour3**.

The **pcolor** function shows the same information that is in a contour plot by mapping the surface height to a set of colors. It is sometimes useful to combine the two on the same plot. MATLAB provides the **contourf** function

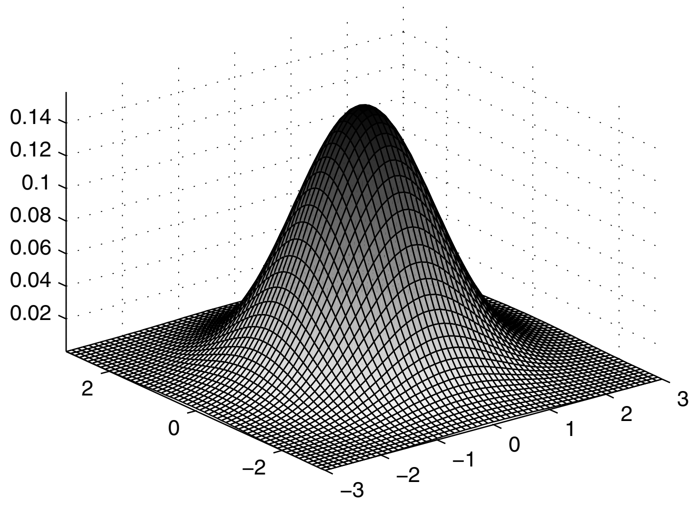

FIGURE 5.18
This shows a **surf** plot of a bivariate normal probability density function.

that will create a combination **pcolor** and **contour** plot. The various options that are available for creating contour plots are illustrated in Example 5.13.

Example 5.13

MATLAB has a function called **peaks** that returns a surface with peaks and depressions that can be used to illustrate contour plots. We show how to use the **peaks** function in this example. The following MATLAB code demonstrates how to create the 2-D contour plot in Figure 5.19.

```
% Get the data for plotting.
[x,y,z] = peaks;
% Create a 2-D contour plot with labels.
% This returns the information for the labels.
c = contour(x,y,z);
% Add the labels to the plot.
clabel(c)
```

A filled contour plot, which is a combination of **pcolor** and **contour**, is given in Figure 5.20. The MATLAB command needed to get this plot is given here.

```
% Create a 2-D filled contour plot.
contourf(x,y,z,15)
```

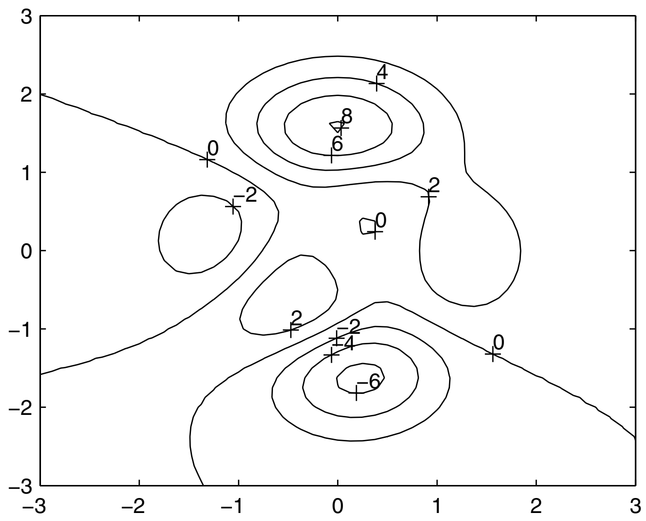

FIGURE 5.19
This is a labeled contour plot of the **peaks** function. The labels make it easier to understand the hills and valleys in the surface.

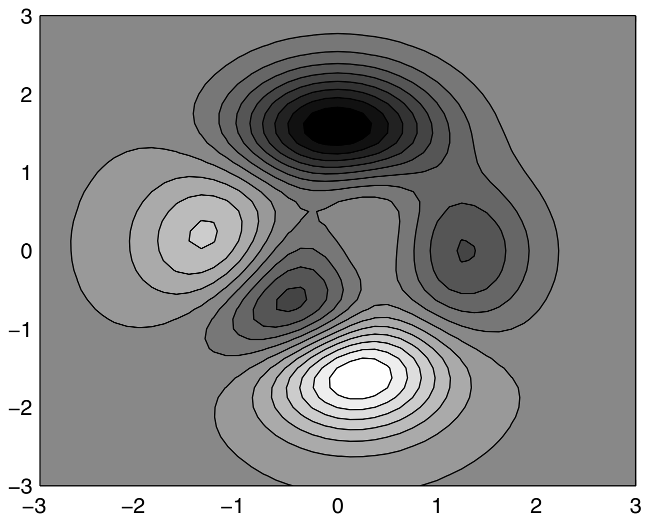

FIGURE 5.20
This is a filled contour plot of the **peaks** surface. It is created using the **contourf** function.

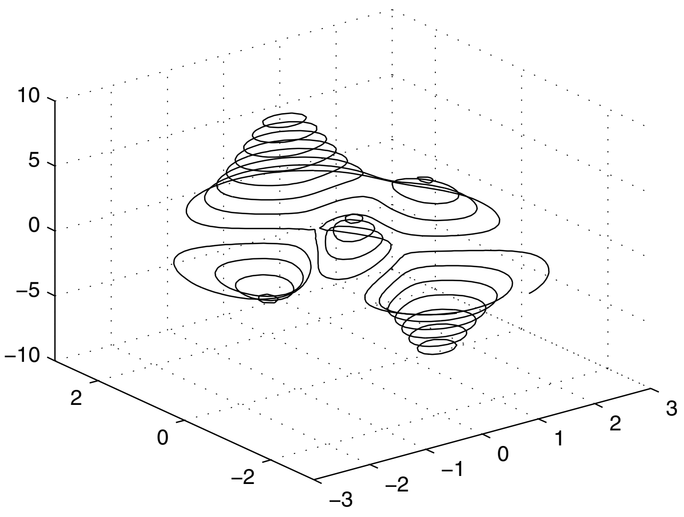

FIGURE 5.21
This is a 3-D contour plot of the **peaks** function.

Finally, a 3-D contour plot is easily obtained using the **contour3** function as shown below. The resulting contour plot is shown in Figure 5.21.

```
% Create a 3-D contour plot.
contour3(x,y,z,15)
```

❑

Bivariate Histogram

In the last section, we described the univariate density histogram as a way of viewing how our data are distributed over the range of the data. We can extend this to any number of dimensions over a partition of the space [Scott, 1992]. However, in this section we restrict our attention to the bivariate histogram given by

$$\hat{f}(\mathbf{x}) = \frac{\nu_k}{nh_1h_2} \qquad \mathrm{x\ in\ } B_k, \tag{5.7}$$

where ν_k represents the number of observations falling into the bivariate bin B_k and h_i is the width of the bin for the x_i coordinate axis. Example 5.14 shows how to get the bivariate density histogram in MATLAB.

Example 5.14

We generate bivariate standard normal random variables and use them to illustrate how to get the bivariate density histogram. We use the optimal bin width for data generated from a standard bivariate normal given in Scott [1992]. We postpone discussion of the optimal bin width and how to obtain it until Chapter 8. A scatterplot of the data and the resulting histogram are shown in Figure 5.22.

```
% Generate sample that is
% standard normal in each dimension.
n = 1000;
d = 2;
x = randn(n,d);
% Need bin origins.
bin0 = [floor(min(x(:,1))) floor(min(x(:,2)))];
% The bin widths - h - are covered later.
h = 3.504*n^(-0.25)*ones(1,2);
% find the number of bins
nb1 = ceil((max(x(:,1))-bin0(1))/h(1));
nb2 = ceil((max(x(:,2))-bin0(2))/h(2));
% find the mesh
t1 = bin0(1):h(1):(nb1*h(1)+bin0(1));
t2 = bin0(2):h(2):(nb2*h(2)+bin0(2));
[X,Y] = meshgrid(t1,t2);
% Find bin frequencies.
[nr,nc] = size(X);
vu = zeros(nr-1,nc-1);
for i = 1:(nr-1)
   for j = 1:(nc-1)
      xv = [X(i,j) X(i,j+1) X(i+1,j+1) X(i+1,j)];
      yv = [Y(i,j) Y(i,j+1) Y(i+1,j+1) Y(i+1,j)];
      in = inpolygon(x(:,1),x(:,2),xv,yv);
      vu(i,j) = sum(in(:));
   end
end
Z = vu/(n*h(1)*h(2));
% Get some axes that make sense.
[XX,YY] = meshgrid(linspace(-3,3,nb1),...
     linspace(-3,3,nb2));
surf(XX,YY,Z)
```

❑

We displayed the resulting bivariate histogram using the **surf** plot in MATLAB. The matrix **Z** in Example 5.14 contains the bin heights. When MATLAB constructs a **mesh** or **surf** plot, the elements of the **Z** matrix represent heights above the x-y plane. The surface is obtained by plotting the

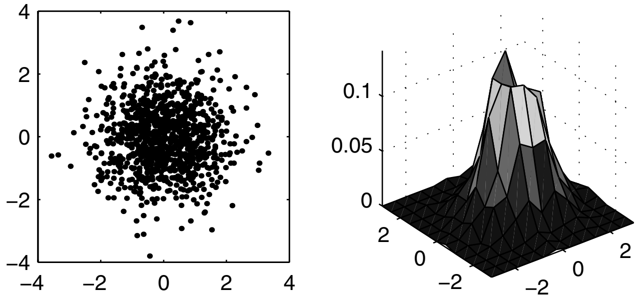

FIGURE 5.22
On the left is a scatterplot of the data. A surface plot of the bivariate density histogram is on the right. Compare the estimated density given by the surface with the one shown in Figure 5.18.

points and joining adjacent points with straight lines. Therefore, a **`surf`** or **`mesh`** plot of the bivariate histogram bin heights is a linear interpolation between adjacent bins. In essence, it provides a smooth version of a histogram. In the next example, we offer another method for viewing the bivariate histogram.

Example 5.15

In this example, we show the bin heights of the bivariate histogram as bars using the MATLAB function `bar3`. The colors are mapped to the column number of the `Z` matrix, not to the heights of the bins. The resulting histogram is shown in Figure 5.23.

```
% The Z matrix is obtained in Example 5.14.
bar3(Z,1)
% Use some Handle Graphics.
set(gca,'YTickLabel',' ','XTickLabel',' ')
set(gca,'YTick',0,'XTick',0)
grid off
```

The following MATLAB code constructs a plot that displays the distribution in a different way. We can use the `scatter` plotting function with arguments

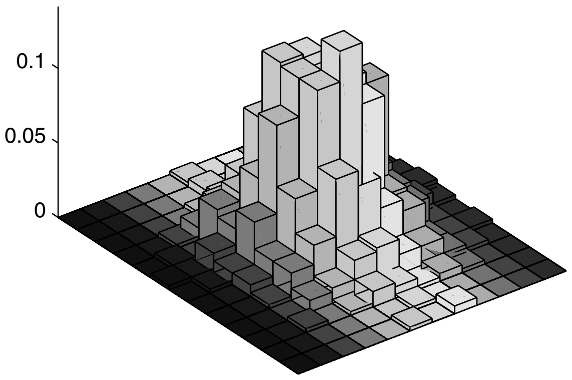

FIGURE 5.23
This shows the same bivariate histogram of Figure 5.22, where the heights of the bars are plotted using the MATLAB function **`bar3`**.

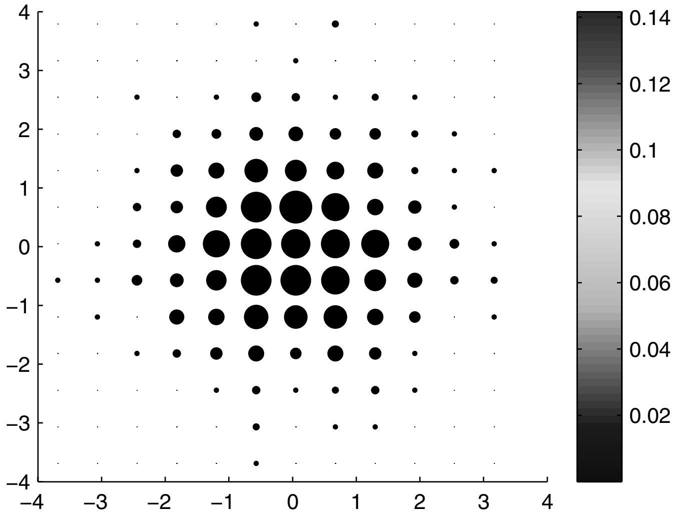

FIGURE 5.24
Here is a different display of the bivariate histogram of Example 5.15. The size and color of the markers indicate the heights of the bins.

that relate the marker size and color to the height of the bins. We add the **colorbar** to map the heights of the bins to the color.

```
% Plot the 2-D histogram as a scatterplot with
% heights proportional to marker size.
% Find the bin centers to use in the scatterplot.
n1 = length(t1);
n2 = length(t2);
tt1 = linspace((t1(1)+t1(2))/2,...
   (t1(n1-1)+t1(n1))/2,nb1);
tt2 = linspace((t2(1)+t2(2))/2,...
   (t2(n2-1)+t2(n2))/2,nb2);
[xxs,yys] = meshgrid(tt1,tt2);
scatter(xxs(:),yys(:),(Z(:)+eps)*1000,...
     (Z(:)+eps)*1000,'filled')
% Create a colorbar and set the axis
% to the correct scale
h_ax = colorbar;
% Get the current labels.
temp = get(h_ax,'Yticklabel');
[nr,nc] = size(temp);
% Convert from strings to numbers.
newlab = cell(nr,1);
tempcell = cellstr(temp);
% Re-scale and convert back to numbers.
for i=1:nr
   newlab{i}=num2str((str2num(tempcell{i})/1000));
end
set(h_ax,'Yticklabel',newlab)
```

This graphic is given in Figure 5.24. Note that we still see the same bivariate normal distribution. The reader might want to compare this plot with the scatterplot of the sample shown in Figure 5.22.
❑

3-D Scatterplot

As with 2-D data, one way we can view trivariate data is with the scatterplot. This is the 3-D analog of the bivariate scatterplot. In this case, the ordered triples (x, y, z) are plotted as points. MATLAB provides a function called **scatter3** that will create a 3-D scatterplot. Analogous to the bivariate case, you can also use the **plot3** function using a symbol for the marker style to obtain a 3-D scatterplot.

A useful MATLAB command when visualizing anything in 3-D is **rotate3d**. Simply type this in at the command line, and you will be able to rotate your graphic using the mouse. There is also a toolbar button that activates the same capability. One reason for looking at scatterplots of the data is to look for interesting structures. The ability to view these structures for 3-D data is dependent on the viewpoint or projection to the screen. When looking at 3-D scatterplots, the analyst should rotate them to search the data for patterns or structure.

Example 5.16

Three variables were measured on ten insects from each of three species [Hand, et al.,1994]. The variables correspond to the width of the first joint of the first tarsus, the width of the first joint of the second tarsus and the maximal width of the aedeagus. All widths are measured in microns. These data were originally used in cluster analysis [Lindsey, Herzberg, and Watts, 1987]. What we would like to see from the scatterplot is whether the data for each species can be separated from the others. In other words, is there clear separation or clustering between the species using these variables? The 3-D scatterplot for these data is shown in Figure 5.25. This view of the scatterplot indicates that using these variables for pattern recognition or clustering (see Chapter 9) is reasonable.

```
% Load the insect data
load insect
% Create a 3-D scatter plot using a
% different color and marker
% for each class of insect.
% Plot the first class and hold the plot.
plot3(insect(1:10,1),insect(1:10,2),...
   insect(1:10,3),'ro')
hold on
% Plot the second class.
plot3(insect(11:20,1),insect(11:20,2),...
   insect(11:20,3),'gx')
% Plot the third class.
plot3(insect(21:30,1),insect(21:30,2),...
   insect(21:30,3),'b*')
% Be sure to turn the hold off!
hold off
```

❑

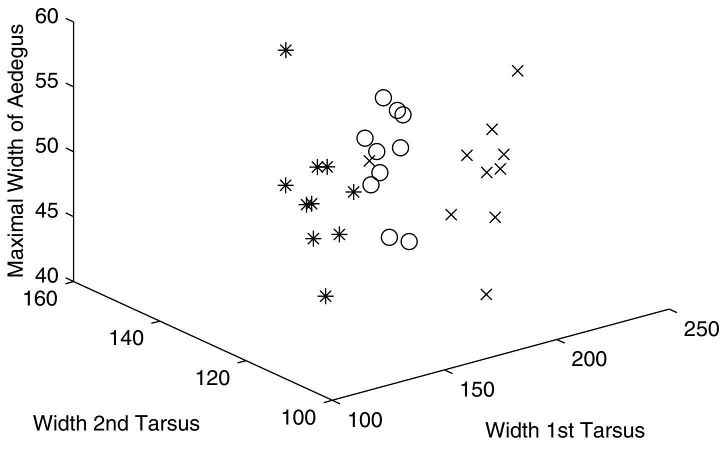

FIGURE 5.25
This is a 3-D scatterplot of the `insect` data. Each species is plotted using a different symbol. This plot indicates that we should be able to identify (with reasonable success) the species based on these three variables.

5.4 Exploring Multi-Dimensional Data

Several methods have been developed to address the problem of visualizing multi-dimensional data. Here we consider applications where we are trying to explore data that has more than three dimensions $(d > 3)$.

We discuss several ways of statically visualizing multi-dimensional data. These include the scatterplot matrix, slices, 3-D contours, star plots, Andrews curves, and parallel coordinates. We finish this section with a description of projection pursuit exploratory data analysis and the grand tour. The grand tour provides a dynamic display of projections of multi-dimensional data, and projection pursuit looks for structure in 1-D or 2-D projections. It should be noted that some of the methods presented here are not restricted to the case where the dimensionality of our data is greater than 3-D.

Scatterplot Matrix

In the previous sections, we presented the scatterplot as a way of looking at 2-D and 3-D data. We can extend this to multi-dimensional data by looking at 2-D scatterplots of all possible pairs of variables. This allows one to view pairwise relationships and to look for interesting structures in two dimensions. MATLAB provides a function called **`plotmatrix`** that will create a scatterplot matrix. Its use is illustrated below.

Example 5.17

The **`iris`** data are well-known to statisticians and are often used to illustrate classification, clustering or visualization techniques. The data were collected by Anderson [1935] and were analyzed by Fisher [1936], so the data are often called *Fisher's iris data* by statisticians. The data consist of 150 observations containing four measurements based on the petals and sepals of three species of iris. These three species are: *Iris setosa, Iris virginica* and *Iris versicolor.* We apply the **`plotmatrix`** function to the iris data set.

```
load iris
% This loads up three matrices, one for each species.
% Get the plotmatrix display of the Iris setosa data.
[H,ax,bigax,P] = plotmatrix(setosa);
axes(bigax),title('Iris Setosa')
```

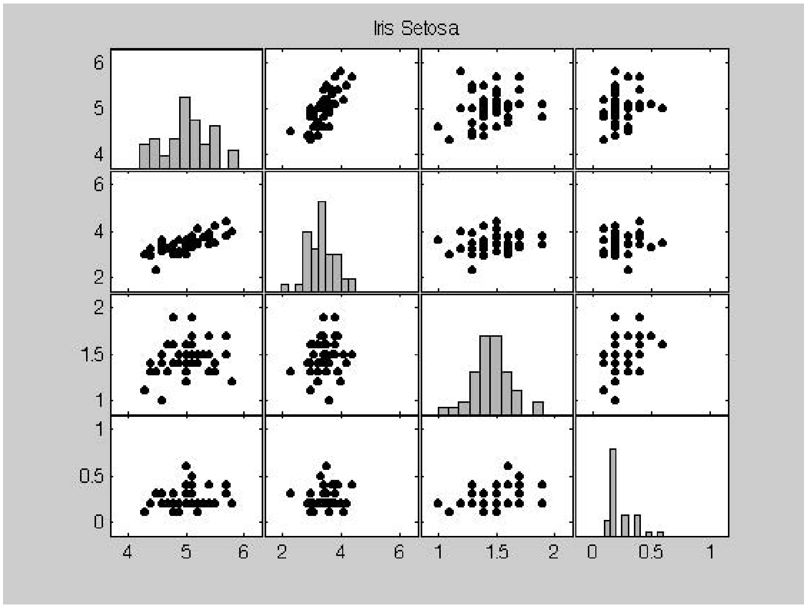

FIGURE 5.26
This is the scatterplot matrix for the *Iris setosa* data using the **`plotmatrix`** function.

The results are shown in Figure 5.26. Several argument options are available for the **plotmatrix** function. If the first two arguments are matrices, then MATLAB plots one column versus the other column. In our example, we use a single matrix argument, and MATLAB creates scatterplots of all possible pairs of variables. Histograms of each variable or column are shown along the diagonal of the scatterplot matrix. Optional output arguments allow one to add a title or change the plot as shown in the following MATLAB commands. Here we replace the histograms with text that identifies the variable names and display the result in Figure 5.27.

```
% Create the labels as a cell array of strings.
labs = {'Sepal Length','Sepal Width',...
     'Petal Length', 'Petal Width'};
[H,ax,bigax,P] = plotmatrix(virginica);
axes(bigax)
title('Virginica')
% Delete the histograms.
delete(P)
%Put the labels in - the positions might have
% to be adjusted depending on the text.
for i = 1:4
   txtax = axes('Position',get(ax(i,i),'Position'),...
       'units','normalized');
   text(.1, .5,labs{i})
   set(txtax,'xtick',[],'ytick',[],...
       'xgrid','off','ygrid','off','box','on')
end
```

❑

Slices and Isosurfaces

If we have a function defined over a volume, $f(x, y, z)$, then we can view it using the MATLAB **slice** function or the **isosurface** function (available in MATLAB 5.3 and higher). This situation could arise in cases where we have a probability density function defined over a volume. The **slice** capability allows us to view the distribution of our data on slices through a volume. The **isosurface** function allows us to view 3-D contours through our volume. These are illustrated in the following examples.

Example 5.18

To illustrate the **slice** function, we need $f(x, y, z)$ values that are defined over a 3-D grid or volume. We will use a trivariate normal distribution centered at the origin with covariance equal to the identity matrix. The following MATLAB code displays slices through the $x = 0$, $y = 0$, and $z = 0$ planes, and the resulting display is shown in Figure 5.28. A standard normal bivari-

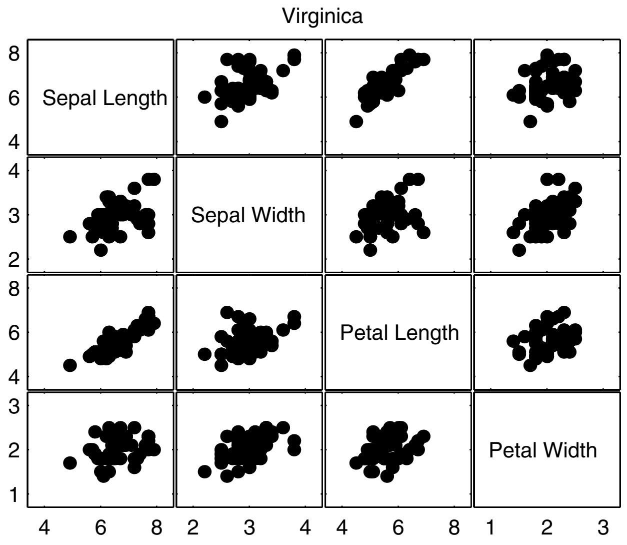

FIGURE 5.27
By using MATLAB's Handle Graphics, we can add text for the variable name to the diagonal boxes.

ate density is given in Figure 5.29 to help the reader understand what the **slice** function is showing. The density or height of the surface defined over the volume is mapped to a color. Therefore, in the **slice** plot, you can see that the maximum density or surface height is at the origin with the height decreasing at the edges of the slices. The color at each point is obtained by interpolation into the volume $f(x, y, z)$.

```
% Create a grid for the domain.
[x,y,z] = meshgrid(-3:.1:3,-3:.1:3,-3:.1:3);
[n,d] = size(x(:));
% Evaluate the trivariate standard normal.
a = (2*pi)^(3/2);
arg = (x.^2 + y.^2 + z.^2);
prob = exp((-.5)*arg)/a;
% Slice through the x=0, y=0, z=0 planes.
slice(x,y,z,prob,0,0,0)
xlabel('X Axis'),ylabel('Y Axis'),zlabel('Z Axis')
```

❑

Isosurfaces are a way of viewing contours through a volume. An isosurface is a surface where the function values $f(x, y, z)$ are constant. These are similar to α-level contours [Scott, 1992], which are defined by

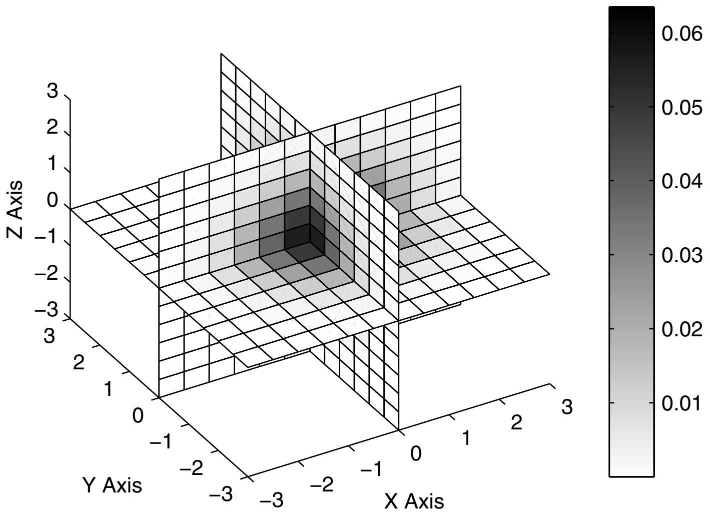

FIGURE 5.28
These are slices through the $x = 0, y = 0, z = 0$ planes for a standard trivariate normal distribution. Each of these planes slice through the volume, and the value of the volume (in this case, the height of the trivariate normal density) is represented by the color. The mode at the origin is clearly seen. We can also see that it is symmetric, because the volume is a mirror image in every slice. Finally, note that the ranges for all the axes are consistent with a standard normal distribution.

$$S_\alpha = \{\mathbf{x}\!: f(\mathbf{x}) = \alpha f_{max}\}; \qquad 0 \le \alpha \le 1\,, \tag{5.8}$$

where **x** is a d-dimensional vector. Generally, the α -level contours are nested surfaces.

The MATLAB function **`isosurface(X,Y,Z,V,isosvalue)`** determines the contour from the volume data **`V`** at the value given by **`isovalue`**. The arrays in **`X`**, **`Y`**, and **`Z`** define the coordinates for the volume. The outputs from this function are the faces and vertices corresponding to the isosurface and can be passed directly into the **`patch`** function for displaying.

Example 5.19

We illustrate several isosurfaces of 3-D contours for data that is uniformly distributed over the volume defined by a unit cube. We display two contours of different levels in Figures 5.30 and 5.31.

```
% Get some data that will be between 0 and 1.
data = rand(10,10,10);
data = smooth3(data,'gaussian');
```

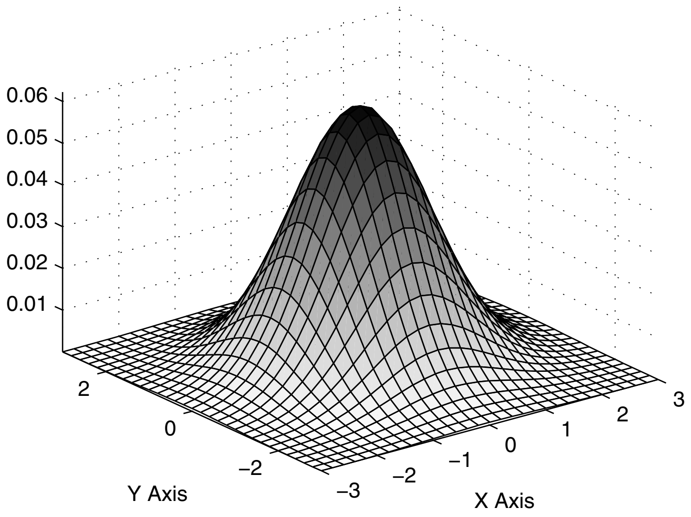

FIGURE 5.29
This is the surface plot for a standard normal bivariate distribution. to help the reader understand what is shown in Figure 5.28.

```
% Just in case there are some figure windows
% open - we should start anew.
close all
for i = [0.4 0.6]
   figure
   hpatch=patch(isosurface(data,i),...
      'Facecolor','blue',...
     'Edgecolor','none',...
     'AmbientStrength',.2,...
     'SpecularStrength',.7,...
     'DiffuseStrength',.4);
   isonormals(data,hpatch)
   title(['f(x,y,z) = ' num2str(i)])
   daspect([1,1,1])
   axis tight
   axis off
   view(3)
   camlight right
   camlight left
   lighting phong
   drawnow
end
```

In Figure 5.30, we have the isosurface for $f(x, y, z) = 0.4$. The isosurface for $f(x, y, z) = 0.6$ is given in Figure 5.31. Again, these are surface contours where the value of the volume is the same.
❑

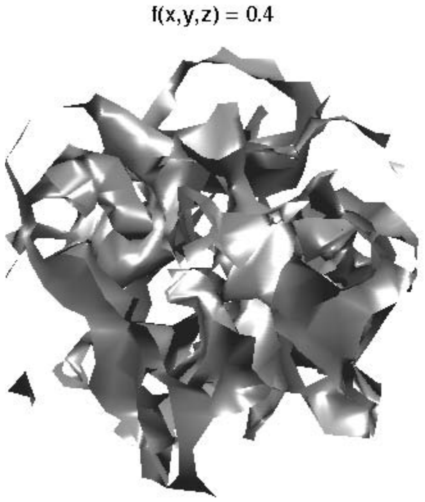

FIGURE 5.30
This is the isosurface of Example 5.19 for $f(x, y, z) = 0.4$.

It would be better if we had a context to help us understand what we are viewing with the isosurfaces. This can be done easily in MATLAB using the function called **`isocaps`**. This function puts caps on the boundaries of the domain and shows the distribution of the volume $f(x, y, z)$ above the isosurface. The color of the cap is mapped to the values $f(x, y, z)$ that are above the given value **`isovalue`**. Values below the **`isovalue`** can be shown on the **`isocap`** via the optional input argument, **`enclose`**. The following example illustrates this concept by adding isocaps to the surfaces obtained in Example 5.19.

Example 5.20

These MATLAB commands show how to add **`isocaps`** to the isosurfaces in the previous example.

```
for i=[0.4 0.6]
 figure
 hpatch = patch(isosurface(data,i),...
   'Facecolor','blue',...
   'Edgecolor','none',...
```

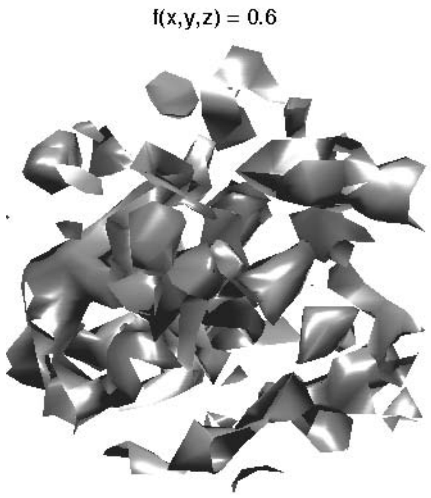

FIGURE 5.31
This is the isosurface of Example 5.19 for $f(x, y, z) = 0.6$.

```
        'AmbientStrength',.2,...
        'SpecularStrength',.7,...
        'DiffuseStrength',.4);
      isonormals(data,hpatch)
      patch(isocaps(data,i),...
        'Facecolor','interp',...
        'EdgeColor','none')
      colormap hsv
      title(['f(x,y,z) = ' num2str(i)])
      daspect([1,1,1])
      axis tight
      axis off
      view(3)
      camlight right
      camlight left
      lighting phong
      drawnow
    end
```

Figure 5.32 shows the **isosurface** of Figure 5.30 with the **isocaps**. It is easier now to see what values are *'inside'* the isosurface or contour. Figure 5.33 shows the **isocaps** added to the **isosurface** corresponding to Figure 5.31.
❑

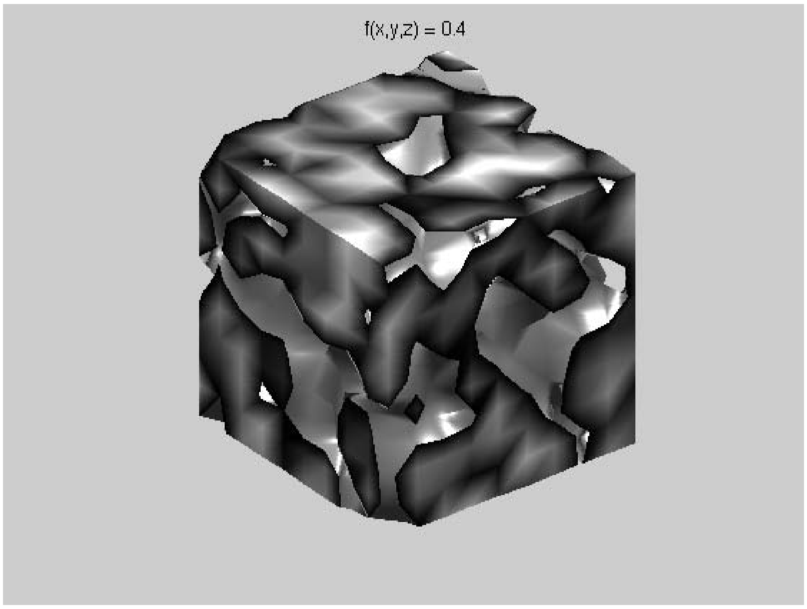

FIGURE 5.32
This is the **isosurface** of Figure 5.30 with **isocaps** added. Note that the color of the edges is mapped to the volume. The default is to map all values above $f(x, y, z) = 0.4$ to the color on the **isocaps**. This can be changed by an input argument to **isocaps**.

Star Plots

Star diagrams were developed by Fienberg [1979] as a way of viewing multidimensional observations as a glyph or star. Each observed data point in the sample is plotted as a star, with the value of each measurement shown as a radial line from a common center point. Thus, each measured value for an observation is plotted as a spoke that is proportional to the size of the measured variable with the ends of the spokes connected with line segments to form a star. Star plots are a nice way to view the entire data set over all dimensions, but they are not suitable when there is a large number of observations ($n > 10$) or many dimensions (e.g., $d > 15$).

The next example applies this technique to data obtained from ratings of eight brands of cereal [Chakrapani and Ehrenberg, 1981; Venables and Ripley, 1994]. In our version of the star plot, the first variable is plotted as the spoke at angle $\theta = 0$, and the rest are shown counter-clockwise from there.

Example 5.21

This example shows the MATLAB code to plot d-dimensional observations in a star plot. The **cereal** file contains a matrix where each row corresponds to

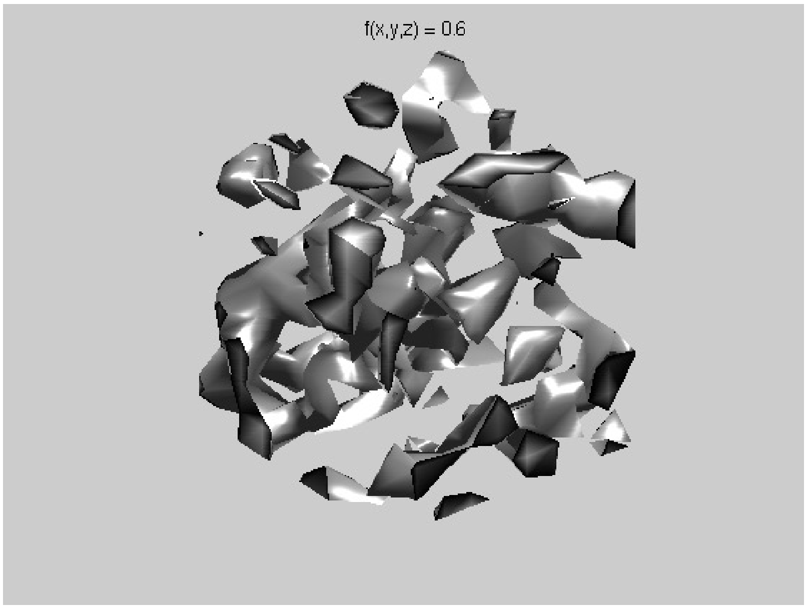

FIGURE 5.33
This is the **isosurface** of Figure 5.31 with **isocaps** added. Note that the color of the edges is mapped to the volume.

an observation and each column represents one of the variables or the percent agreement with the following statements about the cereal:

- come back to
- tastes nice
- popular with all the family
- very easy to digest
- nourishing
- natural flavor
- reasonably priced
- a lot of food value
- stays crispy in milk
- helps to keep you fit
- fun for children to eat

The resulting star plot is shown in Figure 5.34.

```
load cereal
% This file contains the labels and
% the matrix of 8 observations.
```

```
clf
n = 8;
p = 11;
% Find number of rows and columns for the stars.
ncol = floor(sqrt(n));
nrow = ceil(n/ncol);
% Re-scale the data.
md = min(cereal(:));
data = 1 + cereal - md;
% Get angles that are linearly spaced.
% Do not use the last point.
theta = linspace(0,2*pi,p+1);
theta(end) = [];
k = 0;
for i = 1:n
   k = k+1;
   % get the observation for plotting
   r = data(k,:);
   [x,y] = pol2cart(theta,r);
   X = x(:);  % make col vectors
   Y = y(:);
   X = [zeros(p,1) X];
   Y = [zeros(p,1) Y];
   x = [x(:); x(1)];
   y = [y(:); y(1)];
   subplot(nrow,ncol,k),
   patch(x,y,'w')
   hold on
   plot(X(1,:),Y(1,:))
   for ii = 2:p
     plot(X(ii,:),Y(ii,:))
   end
   title(labs{k})
   axis off
   hold off
end
```

❑

Andrews Curves

Andrews curves [Andrews, 1972] were developed as a method for visualizing multi-dimensional data by mapping each observation onto a function. This is similar to star plots in that each observation or sample point is represented by a glyph, except that in this case the glyph is a curve. This function is defined as

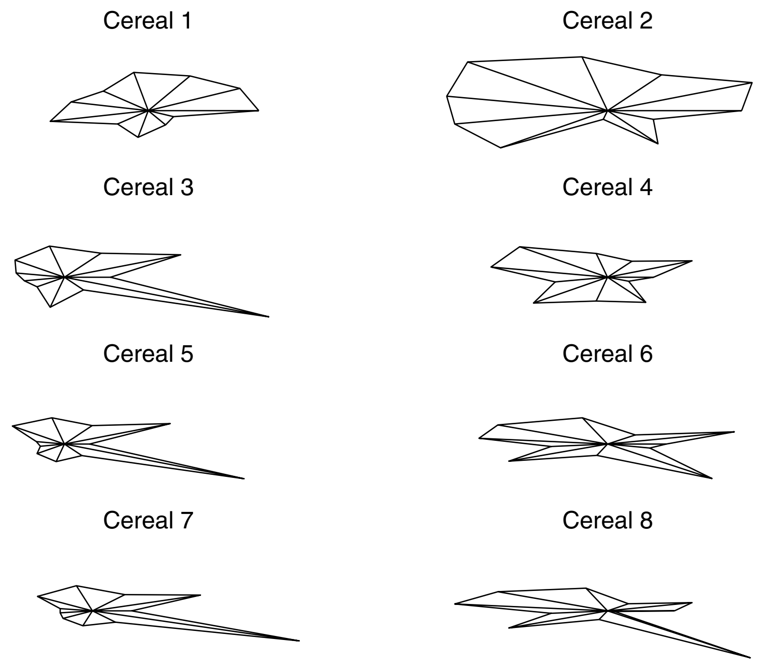

FIGURE 5.34
This is the star plot of the `cereal` data.

$$f_{\mathbf{x}}(t) = x_1/\sqrt{2} + x_2 \sin t + x_3 \cos t + x_4 \sin 2t + x_5 \cos 2t + \dots, \tag{5.9}$$

where the range of t is given by $-\pi \le t \le \pi$. Each observation is projected onto a set of orthogonal basis functions represented by sines and cosines and then plotted. Thus, each sample point is now represented by a curve given by Equation 5.9. We illustrate how to get the Andrews curves in Example 5.22.

Example 5.22

We use a simple example to show how to get Andrews curves. The data we have are the following observations:

$$\mathbf{x}_1 = (2, 6, 4)$$
$$\mathbf{x}_2 = (5, 7, 3)$$
$$\mathbf{x}_3 = (1, 8, 9).$$

Using Equation 5.9, we construct three curves, one corresponding to each data point. The Andrews curves for the data are:

$$f_{\mathbf{x}_1}(t) = 2/\sqrt{2} + 6\sin t + 4\cos t$$

$$f_{\mathbf{x}_2}(t) = 5/\sqrt{2} + 7\sin t + 3\cos t$$

$$f_{\mathbf{x}_3}(t) = 1/\sqrt{2} + 8\sin t + 9\cos t.$$

We can plot these three functions in MATLAB using the following commands. The Andrews curves for these data are shown in Figure 5.35.

```
% Get the domain.
t = linspace(-pi,pi);
% Evaluate function values for each observation.
f1 = 2/sqrt(2)+6*sin(t)+4*cos(t);
f2 = 5/sqrt(2)+7*sin(t)+3*cos(t);
f3 = 1/sqrt(2)+8*sin(t)+9*cos(t);
plot(t,f1,'.',t,f2,'*',t,f3,'o')
legend('F1','F2','F3')
xlabel('t')
```

❑

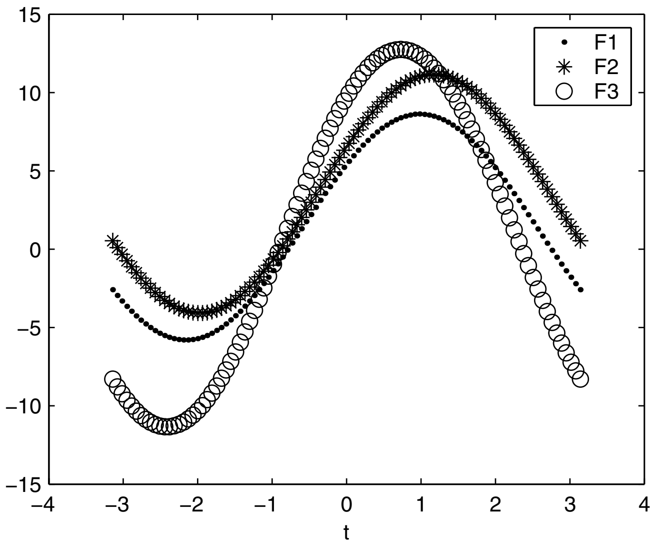

FIGURE 5.35
Andrews curves for the three data points in Example 5.22.

It has been shown [Andrews, 1972; Embrechts and Herzberg, 1991] that because of the mathematical properties of the trigonometric functions, the Andrews curves preserve means, distance (up to a constant) and variances. One consequence of this is that Andrews curves showing functions close together suggest that the corresponding data points will also be close together. Thus, one use of Andrews curves is to look for clustering of the data points.

Example 5.23

We show how to construct Andrews curves for the **iris** data, using only the observations for *Iris setosa* and *Iris virginica* observations. We plot the curves for each species in a different line style to see if there is evidence that we can distinguish between the species using these variables.

```
load iris
% This defines the domain that will be plotted.
theta = (-pi+eps):0.1:(pi-eps);
n = 50;
p = 4;
ysetosa = zeros(n,p);
% There will n curves plotted,
% one for each data point.
yvirginica = zeros(n,p);
% Take dot product of each row with observation.
ang = zeros(length(theta),p);
fstr = '[1/sqrt(2) sin(i) cos(i) sin(2*i)]';
k = 0;
% Evaluate sin and cos functions at each angle theta.
for i = theta
   k = k+1;
   ang(k,:) = eval(fstr);
end
% Now generate a ‘y’ for each observation.
for i = 1:n
  for j = 1:length(theta)
     % Find dot product with observation.
     ysetosa(i,j)=setosa(i,:)*ang(j,:)';
     yvirginica(i,j)=virginica(i,:)*ang(j,:)';
  end
end
% Do all of the plots.
plot(theta,ysetosa(1,:),'r',...
     theta,yvirginica(1,:),'b-.')
legend('Iris Setosa','Iris Virginica')
hold
for i = 2:n
```

```
    plot(theta,ysetosa(i,:),'r',...
       theta,yvirginica(i,:),'b-.')
  end
  hold off
  title('Andrews Plot')
  xlabel('t')
  ylabel('Andrews Curve')
```

The curves are shown in Figure 5.36. By plotting the two groups with different line styles, we can gain some insights about whether or not these two species of iris can be distinguished based on these features. From the Andrews curves, we see that the observations exhibit similarity within each class and that they show differences between the classes. Thus, we might get reasonable discrimination using these features.
❑

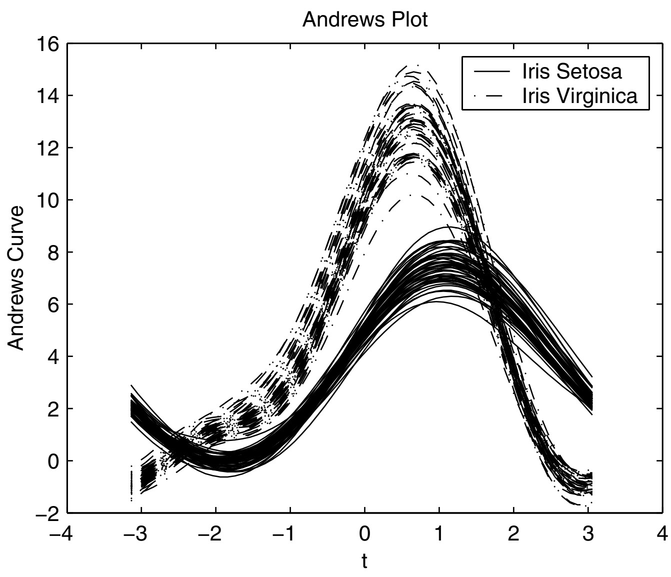

FIGURE 5.36
These are the Andrews curves for the *Iris setosa* and *Iris virginica* data. The curves corresponding to each species are plotted with different line styles. Note that the observations within each group show similar curves, and that we seem to be able to separate these two species.

Andrews curves are dependent on the order of the variables. Lower frequency terms exert more influence on the shape of the curves, so re-ordering the variables and viewing the resulting plot might provide insights about the data. By lower frequency terms, we mean those that are first in the sum given in Equation 5.9. Embrechts and Herzberg [1991] also suggest that the data be rescaled so they are centered at the origin and have covariance equal to the identity matrix. Andrews curves can be extended by using orthogonal bases other than sines and cosines. For example, Embrechts and Herzberg [1991] illustrate Andrews curves using Legendre polynomials and Chebychev polynomials.

Parallel Coordinates

In the Cartesian coordinate system the axes are orthogonal, so the most we can view is three dimensions. If instead we draw the axes parallel to each other, then we can view many axes on the same display. This technique was developed by Wegman [1986] as a way of viewing and analyzing multi-dimensional data and was introduced by Inselberg [1985] in the context of computational geometry and computer vision. Parallel coordinate techniques were expanded on and described in a statistical setting by Wegman [1990]. Wegman [1990] also gave a rigorous explanation of the properties of parallel coordinates as a projective transformation and illustrated the duality properties between the parallel coordinate representation and the Cartesian orthogonal coordinate representation.

A parallel coordinate plot for d-dimensional data is constructed by drawing d lines parallel to each other. We draw d copies of the real line representing the coordinates for $x_1, x_2, \ldots, x_d$. The lines are the same distance apart and are perpendicular to the Cartesian y axis. Additionally, they all have the same positive orientation as the Cartesian x axis. Some versions of parallel coordinates [Inselberg, 1985] draw the parallel axes perpendicular to the Cartesian x axis.

A point $C = (c_1, \ldots, c_4)$ is shown in Figure 5.37 with the MATLAB code that generates it given in Example 5.24. We see that the point is a polygonal line with vertices at $(c_i, i-1), i = 1, \ldots, d$ in Cartesian coordinates on the x_i parallel axis. Thus, a point in Cartesian coordinates is represented in parallel coordinates as a series of connected line segments.

Example 5.24

We now plot the point $C = (1, 3, 7, 2)$ in parallel coordinates using these MATLAB commands.

```
c = [1 3 7 2];
% Get range of parallel axes.
x = [1 7];
% Plot the 4 parallel axes.
plot(x,zeros(1,2),x,ones(1,2),x,...
   2*ones(1,2),x,3*ones(1,2))
hold on
% Now plot point c as a polygonal line.
```

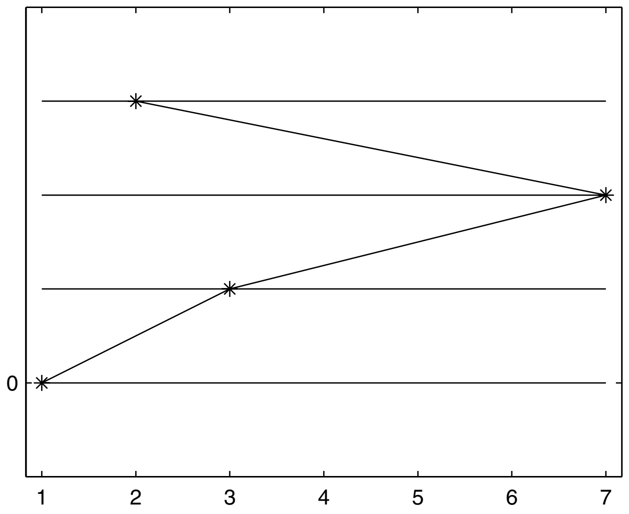

FIGURE 5.37
This shows the parallel coordinate representation for the 4-D point (1,3,7,2).

```
plot(c,0:3,c,0:3,'*')
ax = axis;
axis([ax(1) ax(2) -1 4 ])
set(gca,'ytick',0)
hold off
```

❑

If we plot observations in parallel coordinates with colors designating what class they belong to, then the parallel coordinate display can be used to determine whether or not the variables will enable us to separate the classes. This is similar to the Andrews curves in Example 5.23, where we used the Andrews curves to view the separation between two species of iris. The parallel coordinate plot provides graphical representations of multi-dimensional relationships [Wegman, 1990]. The next example shows how parallel coordinates can display the correlation between two variables.

Example 5.25

We first generate a set of 20 bivariate normal random variables with correlation given by 1. We plot the data using the function called **csparallel** to show how to recognize various types of correlation in parallel coordinate plots.

```
% Get a covariance matrix with correlation 1.
covmat = [1 1; 1 1];
```

```
% Generate the bivariate normal random variables.
% Note: you could use csmvrnd to get these.
[u,s,v] = svd(covmat);
vsqrt = (v*(u'.*sqrt(s)))';
subdata = randn(20,2);
data = subdata*vsqrt;
% Close any open figure windows.
close all
% Create parallel plot using CS Toolbox function.
csparallel(data)
title('Correlation of 1')
```

This is shown in Figure 5.38. The direct linear relationship between the first variable and the second variable is readily apparent. We can generate data that are correlated differently by changing the covariance matrix. For example, to obtain a random sample for data with a correlation of 0.2, we can use

```
covmat = [4 1.2; 1.2, 9];
```

In Figure 5.39, we show the parallel coordinates plot for data that have a correlation coefficient of -1. Note the different structure that is visible in the parallel coordinates plot.
❑

In the previous example, we showed how parallel coordinates can indicate the relationship between variables. To provide further insight, we illustrate how parallel coordinates can indicate clustering of variables in a dimension. Figure 5.40 shows data that can be separated into clusters in both of the dimensions. This is indicated on the parallel coordinate representation by separation or groups of lines along the x_1 and x_2 parallel axes. In Figure 5.41, we have data that are separated into clusters in only one dimension, x_1, but not in the x_2 dimension. This appears in the parallel coordinates plot as a gap in the x_1 parallel axis.

As with Andrews curves, the order of the variables makes a difference. Adjacent parallel axes provide some insights about the relationship between consecutive variables. To see other pairwise relationships, we must permute the order of the parallel axes. Wegman [1990] provides a systematic way of finding all permutations such that all adjacencies in the parallel coordinate display will be visited.

Before we proceed to other topics, we provide an example applying parallel coordinates to the **iris** data. In Example 5.26, we illustrate a parallel coordinates plot of the two classes: *Iris setosa* and *Iris virginica*.

Example 5.26

First we load up the **iris** data. An optional input argument of the **csparallel** function is the line style for the lines. This usage is shown

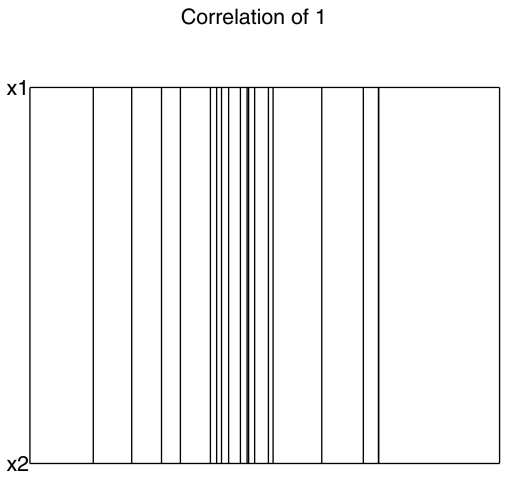

FIGURE 5.38
This is a parallel coordinate plot for bivariate data that have a correlation coefficient of 1.

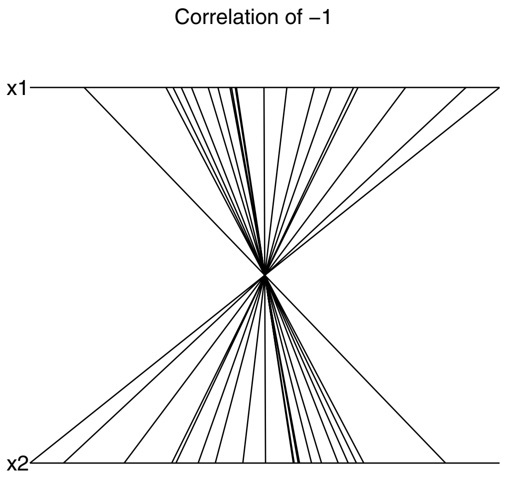

FIGURE 5.39
The data shown in this parallel coordinate plot are negatively correlated.

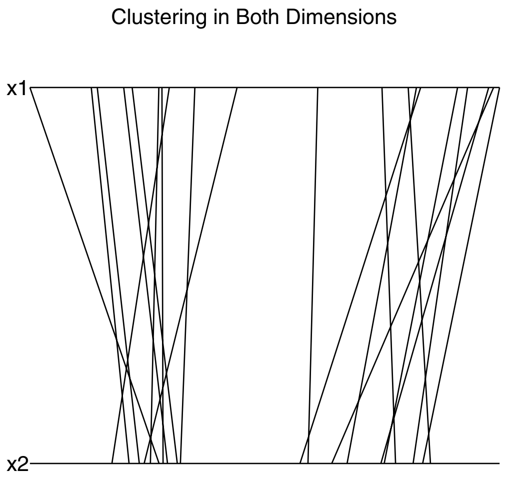

FIGURE 5.40
Clustering in two dimensions produces gaps in both parallel axes.

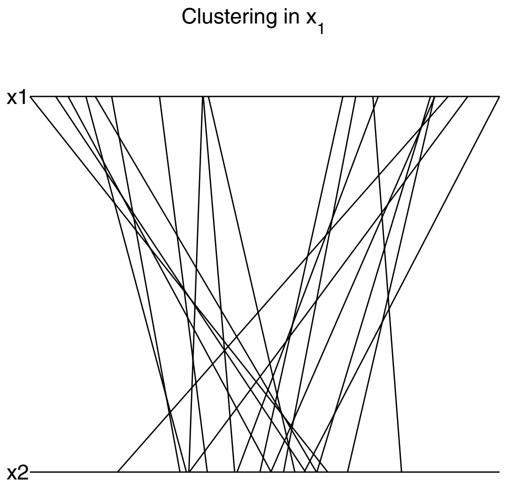

FIGURE 5.41
Clustering in only one dimension produces a gap in the corresponding parallel axis.

below, where we plot the *Iris setosa* observations as dot-dash lines and the *Iris virginica* as solid lines. The parallel coordinate plots is given in Figure 5.42.

```
load iris
figure
csparallel(setosa,'-.')
hold on
csparallel(virginica,'-')
hold off
```

From this plot, we see evidence of groups or separation in coordinates x_2 and x_3.
❑

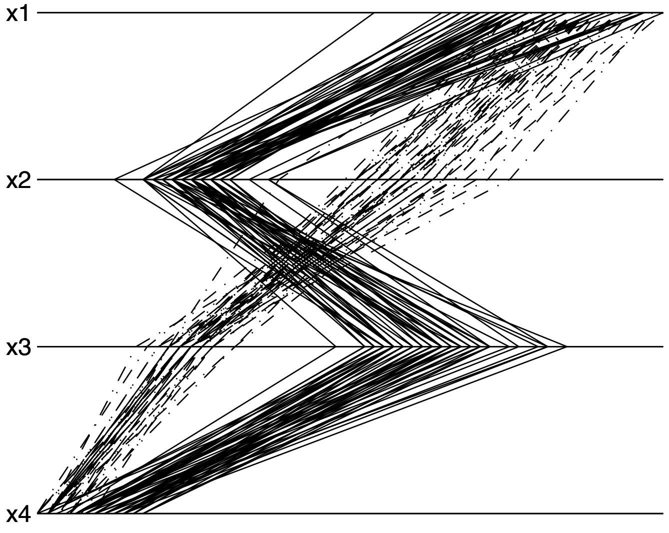

FIGURE 5.42
Here we see an example of a parallel coordinate plot for the **iris** data. The *Iris setosa* is shown as dot-dash lines and the *Iris virginica* as solid lines. There is evidence of groups in two of the coordinate axes, indicating that reasonable separation between these species could be made based on these features.

Projection Pursuit

The Andrews curves and parallel coordinate plots are attempts to visualize all of the data points and all of the dimensions at once. An Andrews curve accomplishes this by mapping a data point to a curve. Parallel coordinate displays accomplish this by mapping each observation to a polygonal line with vertices on parallel axes. Another option is to tackle the problem of visualizing multi-dimensional data by reducing the data to a smaller dimension via a suitable projection. These methods reduce the data to 1-D or 2-D by projecting onto a line or a plane and then displaying each point in some suitable graphic, such as a scatterplot. Once the data are reduced to something that can be easily viewed, then exploring the data for patterns or interesting structure is possible.

One well-known method for reducing dimensionality is ***principal component analysis*** (PCA) [Jackson, 1991]. This method uses the eigenvector decomposition of the covariance (or the correlation) matrix. The data are then projected onto the eigenvector corresponding to the maximum eigenvalue (sometimes known as the first principal component) to reduce the data to one dimension. In this case, the eigenvector is one that follows the direction of the maximum variation in the data. Therefore, if we project onto the first principal component, then we will be using the direction that accounts for the maximum amount of variation using only one dimension. We illustrate the notion of projecting data onto a line in Figure 5.43.

We could project onto two dimensions using the eigenvectors corresponding to the largest and second largest eigenvalues. This would project onto the plane spanned by these eigenvectors. As we see shortly, PCA can be thought of in terms of projection pursuit, where the interesting structure is the variance of the projected data.

There are an infinite number of planes that we can use to reduce the dimensionality of our data. As we just mentioned, the first two principal components in PCA span one such plane, providing a projection such that the variation in the projected data is maximized over all possible 2-D projections. However, this might not be the best plane for highlighting interesting and informative structure in the data. ***Structure*** is defined to be departure from normality and includes such things as clusters, linear structures, holes, outliers, etc. Thus, the objective is to find a projection plane that provides a 2-D view of our data such that the structure (or departure from normality) is maximized over all possible 2-D projections.

We can use the Central Limit Theorem to motivate why we are interested in departures from normality. Linear combinations of data (even Bernoulli data) look normal. Since in most of the low-dimensional projections, one observes a Gaussian, if there is something interesting (e.g., clusters, etc.), then it has to be in the few non-normal projections.

Freidman and Tukey [1974] describe projection pursuit as a way of searching for and exploring nonlinear structure in multi-dimensional data by examining many 2-D projections. The idea is that 2-D orthogonal projections of the

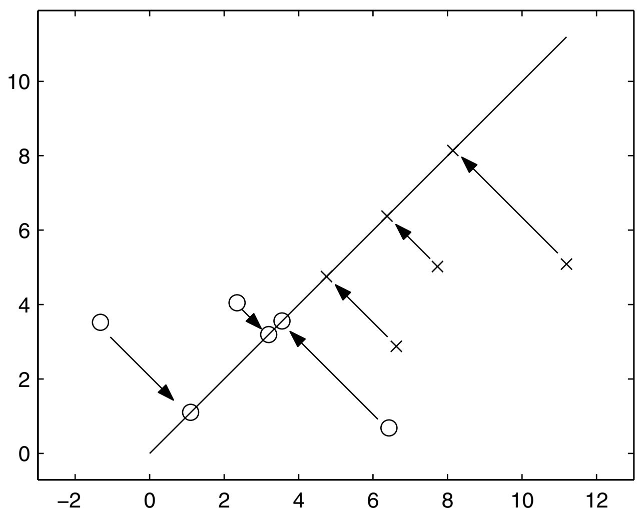

FIGURE 5.43
This illustrates the projection of 2-D data onto a line.

data should reveal structure that is in the original data. The projection pursuit technique can also be used to obtain 1-D projections, but we look only at the 2-D case. Extensions to this method are also described in the literature by Friedman [1987], Posse [1995a, 1995b], Huber [1985], and Jones and Sibson [1987]. In our presentation of projection pursuit exploratory data analysis, we follow the method of Posse [1995a, 1995b].

Projection pursuit exploratory data analysis (PPEDA) is accomplished by visiting many projections to find an interesting one, where *interesting* is measured by an index. In most cases, our interest is in non-normality, so the projection pursuit index usually measures the departure from normality. The index we use is known as the ***chi-square index*** and is developed in Posse [1995a, 1995b]. For completeness, other projection indexes are given in Appendix C, and the interested reader is referred to Posse [1995b] for a simulation analysis of the performance of these indexes.

PPEDA consists of two parts:

1) a projection pursuit index that measures the degree of the structure (or departure from normality), and
2) a method for finding the projection that yields the highest value for the index.

Posse [1995a, 1995b] uses a random search to locate the global optimum of the projection index and combines it with the structure removal of Freidman [1987] to get a sequence of interesting 2-D projections. Each projection found shows a structure that is less important (in terms of the projection index) than the previous one. Before we describe this method for PPEDA, we give a summary of the notation that we use in projection pursuit exploratory data analysis.

NOTATION - PROJECTION PURSUIT EXPLORATORY DATA ANALYSIS

$\mathbf{X}$ is an $n \times d$ matrix, where each row $(\mathbf{X}_i)$ corresponds to a d-dimensional observation and n is the sample size.

$\mathbf{Z}$ is the sphered version of $\mathbf{X}$.

$\hat{\boldsymbol{\mu}}$ is the $1 \times d$ sample mean:

$$\hat{\boldsymbol{\mu}} = \sum \mathbf{X}_i / n . \tag{5.10}$$

$\hat{\boldsymbol{\Sigma}}$ is the sample covariance matrix:

$$\hat{\boldsymbol{\Sigma}}_{ij} = \frac{1}{n-1} \sum (\mathbf{X}_i - \hat{\boldsymbol{\mu}})(\mathbf{X}_j - \hat{\boldsymbol{\mu}})^T . \tag{5.11}$$

α, β are orthonormal ($\alpha^T\alpha = 1 = \beta^T\beta$ and $\alpha^T\beta = 0$) d-dimensional vectors that span the projection plane.

$P(\alpha, \beta)$ is the projection plane spanned by α and β.

$z_i^{\alpha}, z_i^{\beta}$ are the sphered observations projected onto the vectors α and β:

$$\begin{aligned} z_i^{\alpha} &= z_i^T \alpha \\ z_i^{\beta} &= z_i^T \beta \end{aligned} \tag{5.12}$$

(α^*, β^*) denotes the plane where the index is maximum.

$PI_{\chi^2}(\alpha, \beta)$ denotes the chi-square projection index evaluated using the data projected onto the plane spanned by α and β.

ϕ_2 is the standard bivariate normal density.

c_k is the probability evaluated over the k-th region using the standard bivariate normal,

$$c_k = \iint_{B_k} \phi_2 dz_1 dz_2 . \tag{5.13}$$

B_k is a box in the projection plane.

I_{B_k} is the indicator function for region B_k.

$\eta_j = \pi j/36$, $j = 0, \ldots, 8$ is the angle by which the data are rotated in the plane before being assigned to regions B_k.

$\alpha(\eta_j)$ and $\beta(\eta_j)$ are given by

$$\begin{aligned} \alpha(\eta_j) &= \alpha\cos\eta_j - \beta\sin\eta_j \\ \beta(\eta_j) &= \alpha\sin\eta_j + \beta\cos\eta_j \end{aligned} \tag{5.14}$$

c is a scalar that determines the size of the neighborhood around (α^*, β^*) that is visited in the search for planes that provide better values for the projection pursuit index.

v is a vector uniformly distributed on the unit d-dimensional sphere.

half specifies the number of steps without an increase in the projection index, at which time the value of the neighborhood is halved.

m represents the number of searches or random starts to find the best plane.

Projection Pursuit Index

Posse [1995a, 1995b] developed an index based on the chi-square. The plane is first divided into 48 regions or boxes B_k that are distributed in rings. See Figure 5.44 for an illustration of how the plane is partitioned. All regions have the same angular width of 45 degrees and the inner regions have the same radial width of $(2\log 6)^{1/2}/5$. This choice for the radial width provides regions with approximately the same probability for the standard bivariate normal distribution. The regions in the outer ring have probability $1/48$. The regions are constructed in this way to account for the radial symmetry of the bivariate normal distribution.

Posse [1995a, 1995b] provides the population version of the projection index. We present only the empirical version here, because that is the one that must be implemented on the computer. The projection index is given by

$$PI_{\chi^2}(\alpha, \beta) = \frac{1}{9}\sum_{j=1}^{8}\sum_{k=1}^{48}\frac{1}{c_k}\left[\frac{1}{n}\sum_{i=1}^{n} I_{B_k}(z_i^{\alpha(\eta_j)}, z_i^{\beta(\eta_j)}) - c_k\right]^2 . \tag{5.15}$$

The chi-square projection index is not affected by the presence of outliers. This means that an interesting projection obtained using this index will not be one that is interesting solely because of outliers, unlike some of the other indexes (see Appendix C). It is sensitive to distributions that have a hole in the core, and it will also yield projections that contain clusters. The chi-square projection pursuit index is fast and easy to compute, making it appropriate

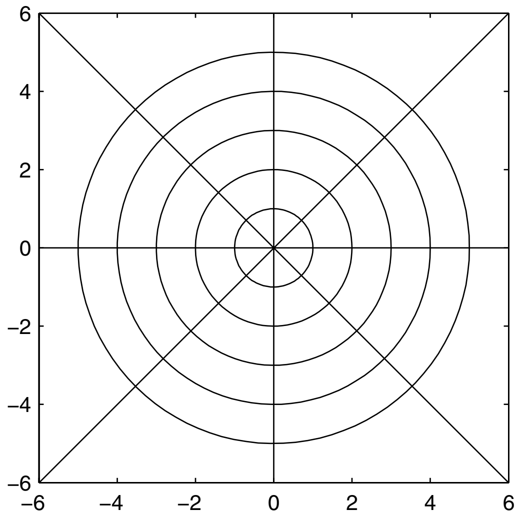

FIGURE 5.44
This shows the layout of the regions B_k for the chi-square projection index. [Posse, 1995a]

for large sample sizes. Posse [1995a] provides a formula to approximate the percentiles of the chi-square index so the analyst can assess the significance of the observed value of the projection index.

Finding the Structure

The second part of PPEDA requires a method for optimizing the projection index over all possible projections onto 2-D planes. Posse [1995a] shows that his optimization method outperforms the steepest-ascent techniques [Friedman and Tukey, 1974]. The Posse algorithm starts by randomly selecting a starting plane, which becomes the current best plane (α^*, β^*). The method seeks to improve the current best solution by considering two candidate solutions within its neighborhood. These candidate planes are given by

$$
\begin{aligned}
a_1 &= \frac{\alpha^* + cv}{\|\alpha^* + cv\|} \qquad & b_1 &= \frac{\beta^* - (a_1^T\beta^*)a_1}{\|\beta^* - (a_1^T\beta^*)a_1\|} \\
a_2 &= \frac{\alpha^* - cv}{\|\alpha^* - cv\|} \qquad & b_1 &= \frac{\beta^* - (a_2^T\beta^*)a_2}{\|\beta^* - (a_2^T\beta^*)a_2\|}.
\end{aligned}
\tag{5.16}
$$

In this approach, we start a global search by looking in large neighborhoods of the current best solution plane (α^*, β^*) and gradually focus in on a maximum by decreasing the neighborhood by half after a specified number of steps with no improvement in the value of the projection pursuit index. When the neighborhood is small, then the optimization process is terminated.

A summary of the steps for the exploratory projection pursuit algorithm is given here. Details on how to implement these steps are provided in Example 5.27 and in Appendix C. The complete search for the best plane involves repeating steps 2 through 9 of the procedure m times, using m random starting planes. Keep in mind that the best plane (α^*, β^*) is the plane where the projected data exhibit the greatest departure from normality.

PROCEDURE - PROJECTION PURSUIT EXPLORATORY DATA ANALYSIS

1. Sphere the data using the following transformation

$$\mathbf{Z}_i = \mathbf{\Lambda}^{-1/2}\mathbf{Q}^T(\mathbf{X}_i - \hat{\boldsymbol{\mu}}) \qquad i = 1, \ldots, n,$$

   where the columns of $\mathbf{Q}$ are the eigenvectors obtained from $\hat{\mathbf{\Sigma}}$, $\mathbf{\Lambda}$ is a diagonal matrix of corresponding eigenvalues, and $\mathbf{X}_i$ is the i-th observation.
2. Generate a random starting plane, (α_0, β_0). This is the current best plane, (α^*, β^*).
3. Evaluate the projection index $PI_{\chi^2}(\alpha_0, \beta_0)$ for the starting plane.
4. Generate two candidate planes (a_1, b_1) and (a_2, b_2) according to Equation 5.16.
5. Evaluate the value of the projection index for these planes, $PI_{\chi^2}(a_1, b_1)$ and $PI_{\chi^2}(a_2, b_2)$.
6. If one of the candidate planes yields a higher value of the projection pursuit index, then that one becomes the current best plane (α^*, β^*).
7. Repeat steps 4 through 6 while there are improvements in the projection pursuit index.
8. If the index does not improve for *half* times, then decrease the value of c by half.
9. Repeat steps 4 through 8 until c is some small number set by the analyst.

Note that in PPEDA we are working with sphered or standardized versions of the original data. Some researchers in this area [Huber, 1985] discuss the benefits and the disadvantages of this approach.

Structure Removal

In PPEDA, we locate a projection that provides a maximum of the projection index. We have no reason to assume that there is only one interesting projection, and there might be other views that reveal insights about our data. To locate other views, Friedman [1987] devised a method called structure removal. The overall procedure is to perform projection pursuit as outlined above, remove the structure found at that projection, and repeat the projection pursuit process to find a projection that yields another maximum value of the projection pursuit index. Proceeding in this manner will provide a sequence of projections providing informative views of the data.

Structure removal in two dimensions is an iterative process. The procedure repeatedly transforms data that are projected to the current solution plane (the one that maximized the projection pursuit index) to standard normal until they stop becoming more normal. We can measure *'more normal'* using the projection pursuit index.

We start with a $d \times d$ matrix $\mathbf{U}^*$, where the first two rows of the matrix are the vectors of the projection obtained from PPEDA. The rest of the rows of $\mathbf{U}^*$ have ones on the diagonal and zero elsewhere. For example, if $d = 4$, then

$$\mathbf{U}^* = \begin{bmatrix} \alpha_1^* & \alpha_2^* & \alpha_3^* & \alpha_4^* \\ \beta_1^* & \beta_2^* & \beta_3^* & \beta_4^* \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We use the Gram-Schmidt process [Strang, 1988] to make $\mathbf{U}^*$ orthonormal. We denote the orthonormal version as $\mathbf{U}$.

The next step in the structure removal process is to transform the $\mathbf{Z}$ matrix using the following

$$\mathbf{T} = \mathbf{U}\mathbf{Z}^T. \tag{5.17}$$

In Equation 5.17, $\mathbf{T}$ is $d \times n$, so each column of the matrix corresponds to a d-dimensional observation. With this transformation, the first two dimensions (the first two rows of $\mathbf{T}$) of every transformed observation are the projection onto the plane given by (α^*, β^*).

We now remove the structure that is represented by the first two dimensions. We let Θ be a transformation that transforms the first two rows of $\mathbf{T}$ to a standard normal and the rest remain unchanged. This is where we actually remove the structure, making the data normal in that projection (the first two rows). Letting $\mathbf{T}_1$ and $\mathbf{T}_2$ represent the first two rows of $\mathbf{T}$, we define the transformation as follows

$$\begin{aligned}\Theta(\mathbf{T}_1) &= \Phi^{-1}[F(\mathbf{T}_1)] \\ \Theta(\mathbf{T}_2) &= \Phi^{-1}[F(\mathbf{T}_2)] \\ \Theta(\mathbf{T}_i) &= \mathbf{T}_i; \qquad i = 3, \ldots, d\ ,\end{aligned} \tag{5.18}$$

where Φ^{-1} is the inverse of the standard normal cumulative distribution function and F is a function defined below (see Equations 5.19 and 5.20). We see from Equation 5.18, that we will be changing only the first two rows of **T**.

We now describe the transformation of Equation 5.18 in more detail, working only with $\mathbf{T}_1$ and $\mathbf{T}_2$. First, we note that $\mathbf{T}_1$ can be written as

$$\mathbf{T}_1 = (z_1^{\alpha^*}, \ldots, z_j^{\alpha^*}, \ldots, z_n^{\alpha^*}),$$

and $\mathbf{T}_2$ as

$$\mathbf{T}_2 = (z_1^{\beta^*}, \ldots, z_j^{\beta^*}, \ldots, z_n^{\beta^*}).$$

Recall that $z_j^{\alpha^*}$ and $z_j^{\beta^*}$ would be coordinates of the j-th observation projected onto the plane spanned by (α^*, β^*).

Next, we define a rotation about the origin through the angle γ as follows

$$\begin{aligned}\tilde{z}_j^{1(t)} &= z_j^{1(t)}\cos\gamma + z_j^{2(t)}\sin\gamma \\ \tilde{z}_j^{2(t)} &= z_j^{2(t)}\cos\gamma - z_j^{1(t)}\sin\gamma,\end{aligned} \tag{5.19}$$

where $\gamma = 0, \pi/4, \pi/8, 3\pi/8$ and $z_j^{1(t)}$ represents the j-th element of $\mathbf{T}_1$ at the t-th iteration of the process. We now apply the following transformation to the rotated points,

$$z_j^{1(t+1)} = \Phi^{-1}\left\{\frac{r(\tilde{z}_j^{1(t)}) - 0.5}{n}\right\} \qquad z_j^{2(t+1)} = \Phi^{-1}\left\{\frac{r(\tilde{z}_j^{2(t)}) - 0.5}{n}\right\}, \tag{5.20}$$

where $r(\tilde{z}_j^{1(t)})$ represents the rank (position in the ordered list) of $\tilde{z}_j^{1(t)}$.

This transformation replaces each rotated observation by its normal score in the projection. With this procedure, we are deflating the projection index by making the data more normal. It is evident in the procedure given below, that this is an iterative process. Friedman [1987] states that during the first few iterations, the projection index should decrease rapidly. After approximate normality is obtained, the index might oscillate with small changes. Usually, the process takes between 5 to 15 complete iterations to remove the structure.

Once the structure is removed using this process, we must transform the data back using

$$\mathbf{Z}' = \mathbf{U}^T \Theta(\mathbf{U}\mathbf{Z}^T) . \qquad (5.21)$$

In other words, we transform back using the transpose of the orthonormal matrix **U**. From matrix theory [Strang, 1988], we see that all directions orthogonal to the structure (i.e., all rows of **T** other than the first two) have not been changed. Whereas, the structure has been Gaussianized and then transformed back.

PROCEDURE - STRUCTURE REMOVAL

1. Create the orthonormal matrix **U**, where the first two rows of **U** contain the vectors α^*, β^*.
2. Transform the data **Z** using Equation 5.17 to get **T**.
3. Using only the first two rows of **T**, rotate the observations using Equation 5.19.
4. Normalize each rotated point according to Equation 5.20.
5. For angles of rotation $\gamma = 0, \pi/4, \pi/8, 3\pi/8$, repeat steps 3 through 4.
6. Evaluate the projection index using $\mathbf{z}_j^{1(t+1)}$ and $\mathbf{z}_j^{2(t+1)}$, after going through an entire cycle of rotation (Equation 5.19) and normalization (Equation 5.20).
7. Repeat steps 3 through 6 until the projection pursuit index stops changing.
8. Transform the data back using Equation 5.21.

Example 5.27

We use a synthetic data set to illustrate the MATLAB functions used for PPEDA. The source code for the functions used in this example is given in Appendix C. These data contain two structures, both of which are clusters. So we will search for two planes that maximize the projection pursuit index. First we load the data set that is contained in the file called **`ppdata`**. This loads a matrix **X** containing 400 six-dimensional observations. We also set up the constants we need for the algorithm.

```
% First load up a synthetic data set.
% This has structure
% in two planes - clusters.
% Note that the data is in
% ppdata.mat
load ppdata
```

```
% For m random starts, find the best projection plane
% using N structure removal procedures.
% Two structures:
N = 2;
% Four random starts:
m = 4;
c = tan(80*pi/180);
% Number of steps with no increase.
half = 30;
```

We now set up some arrays to store the results of projection pursuit.

```
% To store the N structures:
astar = zeros(d,N);
bstar = zeros(d,N);
ppmax = zeros(1,N);
```

Next we have to sphere the data.

```
% Sphere the data.
[n,d] = size(X);
muhat = mean(X);
[V,D] = eig(cov(X));
Xc = X-ones(n,1)*muhat;
Z = ((D)^(-1/2)*V'*Xc')';
```

We use the sphered data as input to the function `csppeda`. The outputs from this function are the vectors that span the plane containing the structure and the corresponding value of the projection pursuit index.

```
% Now do the PPEDA.
% Find a structure, remove it,
% and look for another one.
Zt = Z;
for i = 1:N
   [astar(:,i),bstar(:,i),ppmax(i)] =,...
        csppeda(Zt,c,half,m);
   % Now remove the structure.
   Zt = csppstrtrem(Zt,astar(:,i),bstar(:,i));
end
```

Note that each column of `astar` and `bstar` contains the projections for a structure, each one found using *m* random starts of the Posse algorithm. To see the first structure and second structures, we project onto the best planes as follows:

```
% Now project and see the structure.
proj1 = [astar(:,1), bstar(:,1)];
proj2 = [astar(:,2), bstar(:,2)];
Zp1 = Z*proj1;
```

```
Zp2 = Z*proj2;
figure
plot(Zp1(:,1),Zp1(:,2),'k.'),title('Structure 1')
xlabel('\alpha^*'),ylabel('\beta^*')
figure
plot(Zp2(:,1),Zp2(:,2),'k.'),title('Structure 2')
xlabel('\alpha^*'),ylabel('\beta^*')
```

The results are shown in Figure 5.45 and Figure 5.46, where we see that projection pursuit did find two structures. The first structure has a projection pursuit index of 2.67, and the second structure has an index equal to 0.572.
❑

Grand Tour

The grand tour of Asimov [1985] is an interactive visualization technique that enables the analyst to look for interesting structure embedded in multi-dimensional data. The idea is to project the d-dimensional data to a plane and to rotate the plane through all possible angles, searching for structure in the data. As with projection pursuit, structure is defined as departure from normality, such as clusters, spirals, linear relationships, etc.

In this procedure, we first determine a plane, project the data onto it, and then view it as a 2-D scatterplot. This process is repeated for a sequence of planes. If the sequence of planes is smooth (in the sense that the orientation of the plane changes slowly), then the result is a movie that shows the data points moving in a continuous manner. Asimov [1985] describes two methods for conducting a grand tour, called the ***torus algorithm*** and the ***random interpolation algorithm***. Neither of these methods is ideal. With the torus method we may end up spending too much time in certain regions, and it is computationally intensive. The random interpolation method is better computationally, but cannot be reversed easily (to recover the projection) unless the set of random numbers used to generate the tour is retained. Thus, this method requires a lot of computer storage. Because of these limitations, we describe the ***pseudo grand tour*** described in Wegman and Shen [1993].

One of the important aspects of the torus grand tour is the need for a continuous space-filling path through the manifold of planes. This requirement satisfies the condition that the tour will visit *all* possible orientations of the projection plane. Here, we do not follow a space-filling curve, so this will be called a pseudo grand tour. In spite of this, the pseudo grand tour has many benefits:

- It can be calculated easily;
- It does not spend a lot of time in any one region;
- It still visits an ample set of orientations; and
- It is easily reversible.

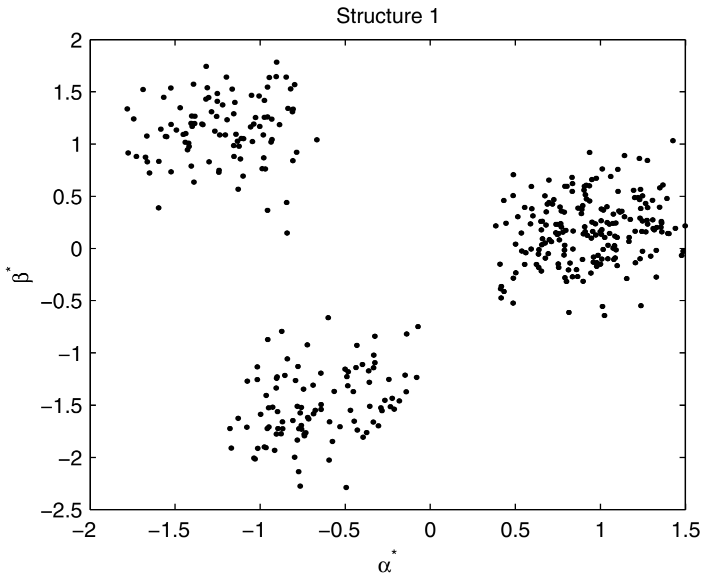

FIGURE 5.45
Here we see the first structure that was found using PPEDA. This structure yields a value of 2.67 for the chi-square projection pursuit index.

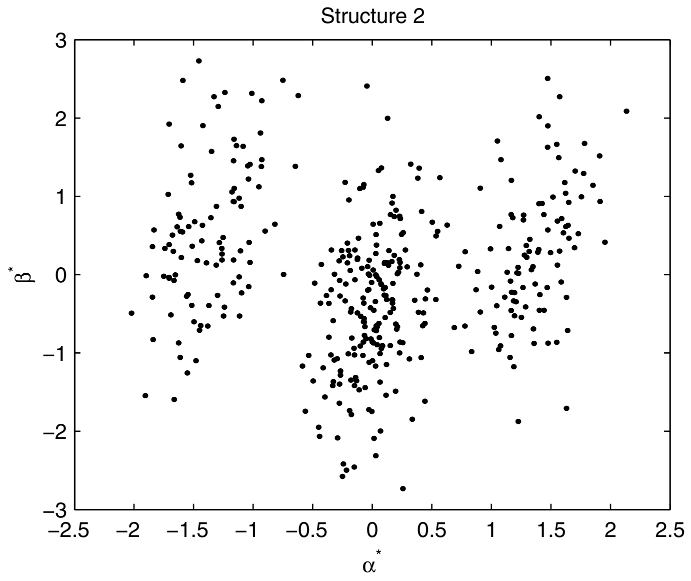

FIGURE 5.46
Here is the second structure we found using PPEDA. This structure has a value of 0.572 for the chi-square projection pursuit index.

The fact that the pseudo grand tour is easily reversible enables the analyst to recover the projection for further analysis. Two versions of the pseudo grand tour are available: one that projects onto a line and one that projects onto a plane.

As with projection pursuit, we need unit vectors that comprise the desired projection. In the 1-D case, we require a unit vector $\boldsymbol{\alpha}(t)$ such that

$$\|\boldsymbol{\alpha}(t)\|^2 = \sum_{i=1}^{d} \alpha_i^2(t) = 1$$

for every t, where t represents a point in the sequence of projections. For the pseudo grand tour, $\boldsymbol{\alpha}(t)$ must be a continuous function of t and should produce *all* possible orientations of a unit vector.

We obtain the projection of the data using

$$z_i^{\boldsymbol{\alpha}(t)} = \boldsymbol{\alpha}^T(t)\mathbf{x}_i, \tag{5.22}$$

where $\mathbf{x}_i$ is the i-th d-dimensional data point. To get the movie view of the pseudo grand tour, we plot $z_i^{\boldsymbol{\alpha}(t)}$ on a fixed 1-D coordinate system, re-displaying the projected points as t increases.

The grand tour in two dimensions is similar. We need a second unit vector $\boldsymbol{\beta}(t)$ that is orthonormal to $\boldsymbol{\alpha}(t)$,

$$\|\boldsymbol{\beta}(t)\|^2 = \sum_{i=1}^{d} \beta_i^2(t) = 1 \qquad \boldsymbol{\alpha}^T(t)\boldsymbol{\beta}(t) = 0.$$

We project the data onto the second vector using

$$z_i^{\boldsymbol{\beta}(t)} = \boldsymbol{\beta}^T(t)\mathbf{x}_i. \tag{5.23}$$

To obtain the movie view of the 2-D pseudo grand tour, we display $z_i^{\boldsymbol{\alpha}(t)}$ and $z_i^{\boldsymbol{\beta}(t)}$ in a 2-D scatterplot, replotting the points as t increases.

The basic idea of the grand tour is to project the data onto a 1-D or 2-D space and plot the projected data, repeating this process many times to provide many views of the data. It is important for viewing purposes to make the time steps small to provide a nearly continuous path and to provide smooth motion of the points. The reader should note that the grand tour is an interactive approach to EDA. The analyst must stop the tour when an interesting projection is found.

Asimov [1985] contends that we are viewing more than one or two dimensions because the speed vectors provide further information. For example, the further away a point is from the computer screen, the faster the point rotates. We believe that the extra dimension conveyed by the speed is difficult to understand unless the analyst has experience looking at grand tour movies.

In order to implement the pseudo grand tour, we need a way of obtaining the projection vectors $\boldsymbol{\alpha}(t)$ and $\boldsymbol{\beta}(t)$. First we consider the data vector **x**. If d is odd, then we augment each data point with a zero, to get an even number of elements. In this case,

$$\mathbf{x} = (x_1, \ldots, x_d, 0); \qquad \text{for } d \text{ odd.}$$

This will not affect the projection. So, without loss of generality, we present the method with the understanding that d is even. We take the vector $\boldsymbol{\alpha}(t)$ to be

$$\boldsymbol{\alpha}(t) = \sqrt{2/d}(\sin\omega_1 t, \cos\omega_1 t, \ldots, \sin\omega_{d/2} t, \cos\omega_{d/2} t) \;, \tag{5.24}$$

and the vector $\boldsymbol{\beta}(t)$ as

$$\boldsymbol{\beta}(t) = \sqrt{2/d}(\cos\omega_1 t, -\sin\omega_1 t, \ldots, \cos\omega_{d/2} t, -\sin\omega_{d/2} t) \,. \tag{5.25}$$

We choose ω_i and ω_j such that the ratio ω_i/ω_j is irrational for every i and j. Additionally, we must choose these such that no ω_i/ω_j is a rational multiple of any other ratio. It is also recommended that the time step Δt be a small positive irrational number. One way to obtain irrational values for ω_i is to let $\omega_i = \sqrt{P_i}$, where P_i is the i-th prime number.

The steps for implementing the 2-D pseudo grand tour are given here. The details on how to implement this in MATLAB are given in Example 5.28.

PROCEDURE - PSEUDO GRAND TOUR

1. Set each ω_i to an irrational number.
2. Find vectors $\boldsymbol{\alpha}(t)$ and $\boldsymbol{\beta}(t)$ using Equations 5.24 and 5.25.
3. Project the data onto the plane spanned by these vectors using Equations 5.23 and 5.24.
4. Display the projected points, $z_i^{\boldsymbol{\alpha}(t)}$ and $z_i^{\boldsymbol{\beta}(t)}$, in a 2-D scatterplot.
5. Using Δt irrational, increment the time, and repeat steps 2 through 4.

Before we illustrate this in an example, we note that once we stop the tour at an interesting projection, we can easily recover the projection by knowing the time step.

Example 5.28

In this example, we use the **iris** data to illustrate the grand tour. First we load up the data and set up some preliminaries.

```
% This is for the iris data.
load iris
% Put data into one matrix.
x = [setosa;virginica;versicolor];
% Set up vector of frequencies.
th = sqrt([2 3]);
% Set up other constants.
[n,d] = size(x);
% This is a small irrational number:
delt = eps*10^14;
% Do the tour for some specified time steps.
maxit = 1000;
cof = sqrt(2/d);
% Set up storage space for projection vectors.
a = zeros(d,1);
b = zeros(d,1);
z = zeros(n,2);
```

We now do some preliminary plotting, just to get the handles we need to use MATLAB's Handle Graphics for plotting. This enables us to update the points that are plotted rather than replotting the entire figure.

```
% Get an initial plot, so the tour can be implemented
% using Handle Graphics.
Hlin1 = plot(z(1:50,1),z(1:50,2),'ro');
set(gcf,'backingstore','off')
set(gca,'Drawmode','fast')
hold on
Hlin2 = plot(z(51:100,1),z(51:100,2),'go');
Hlin3 = plot(z(101:150,1),z(101:150,2),'bo');
hold off
axis equal
axis vis3d
axis off
```

Now we do the actual pseudo grand tour, where we use a maximum number of iterations given by `maxit`.

```
for t = 0:delt:(delt*maxit)
 % Find the transformation vectors.
 for j = 1:d/2
   a(2*(j-1)+1) = cof*sin(th(j)*t);
   a(2*j) = cof*cos(th(j)*t);
   b(2*(j-1)+1) = cof*cos(th(j)*t);
```

```
      b(2*j) = cof*(-sin(th(j)*t));
    end
    % Project onto the vectors.
    z(:,1) = x*a;
    z(:,2) = x*b;
    set(Hlin1,'xdata',z(1:50,1),'ydata',z(1:50,2))
    set(Hlin2,'xdata',z(51:100,1),'ydata',z(51:100,2))
    set(Hlin3,'xdata',z(101:150,1),'ydata',z(101:150,2))
    drawnow
   end
```

❑

5.5 MATLAB Code

MATLAB has many functions for visualizing data, both in the main package and in the Statistics Toolbox. Many of these were mentioned in the text and are summarized in Appendix E. Basic MATLAB has functions for scatterplots (**scatter**), histograms (**hist**, **bar**), and scatterplot matrices (**plotmatrix**). The Statistics Toolbox has functions for constructing q-q plots (**normplot**, **qqplot**, **weibplot**), the empirical cumulative distribution function (**cdfplot**), grouped versions of plots (**gscatter**, **gplotmatrix**), and others. Some other graphing functions in the standard MATLAB package that might be of interest include pie charts (**pie**), stair plots (**stairs**), error bars (**errorbar**), and stem plots (**stem**).

The methods for statistical graphics described in Cleveland's *Visualizing Data* [1993] have been implemented in MATLAB. They are available for download at

http://www.datatool.com/Dataviz_home.htm.

This book contains many useful techniques for visualizing data. Since MATLAB code is available for these methods, we urge the reader to refer to this highly readable text for more information on statistical visualization.

Rousseeuw, Ruts and Tukey [1999] describe a bivariate generalization of the univariate boxplot called a ***bagplot***. This type of plot displays the location, spread, correlation, skewness and tails of the data set. Software (MATLAB and S-Plus®) for constructing a bagplot is available for download at

http://win-www.uia.ac.be/u/statis/index.html.

In the Computational Statistics Toolbox, we include several functions that implement some of the algorithms and graphics covered in Chapter 5. These are summarized in Table 5.3.

TABLE 5.3

List of Functions from Chapter 5 Included in the Computational Statistics Toolbox

Purpose	MATLAB Function
Star Plot	`csstars`
Stem-and-leaf Plot	`csstemleaf`
Parallel Coordinates Plot	`csparallel`
Q-Q Plot	`csqqplot`
Poissonness Plot	`cspoissplot`
Andrews Curves	`csandrews`
Exponential Probability Plot	`csexpoplot`
Binomial Plot	`csbinoplot`

PPEDA	`csppeda`
	`csppstrtrem`
	`csppind`

5.6 Further Reading

One of the first treatises on graphical exploratory data analysis is John Tukey's *Exploratory Data Analysis* [1977]. In this book, he explains many aspects of EDA, including smoothing techniques, graphical techniques and others. The material in this book is practical and is readily accessible to readers with rudimentary knowledge of data analysis. Another excellent book on this subject is *Graphical Exploratory Data Analysis* [du Toit, Steyn and Stumpf, 1986], which includes several techniques (e.g., Chernoff faces and profiles) that we do not cover. For texts that emphasize the visualization of technical data, see Fortner and Meyer [1997] and Fortner [1995]. The paper by Wegman, Carr and Luo [1993] discusses many of the methods we present, along with others such as stereoscopic displays, generalized nonlinear regression using skeletons and a description of d-dimensional grand tour. This paper and Wegman [1990] provide an excellent theoretical treatment of parallel coordinates.

The Grammar of Graphics by Wilkinson [1999] describes a foundation for producing graphics for scientific journals, the internet, statistical packages, or any visualization system. It looks at the rules for producing pie charts, bar charts scatterplots, maps, function plots, and many others.

For the reader who is interested in visualization and information design, the three books by Edward Tufte are recommended. His first book, *The Visual Display of Quantitative Information* [Tufte, 1983], shows how to depict numbers. The second in the series is called *Envisioning Information* [Tufte, 1990], and illustrates how to deal with pictures of nouns (e.g., maps, aerial photographs, weather data). The third book is entitled *Visual Explanations* [Tufte, 1997], and it discusses how to illustrate pictures of verbs. These three books also provide many examples of good graphics and bad graphics. We highly recommend the book by Wainer [1997] for any statistician, engineer or data analyst. Wainer discusses the subject of good and bad graphics in a way that is accessible to the general reader.

Other techniques for visualizing multi-dimensional data have been proposed in the literature. One method introduced by Chernoff [1973] represents d-dimensional observations by a cartoon face, where features of the face reflect the values of the measurements. The size and shape of the nose, eyes, mouth, outline of the face and eyebrows, etc. would be determined by the value of the measurements. Chernoff faces can be used to determine simple trends in the data, but they are hard to interpret in most cases.

Another graphical EDA method that is often used is called brushing. Brushing [Venables and Ripley, 1994; Cleveland, 1993] is an interactive technique where the user can highlight data points on a scatterplot and the same points are highlighted on all other plots. For example, in a scatterplot matrix, highlighting a point in one plot shows up as highlighted in all of the others. This helps illustrate interesting structure across plots.

High-dimensional data can also be viewed using color histograms or data images. Color histograms are described in Wegman [1990]. Data images are discussed in Minotte and West [1998] and are a special case of color histograms.

For more information on the graphical capabilities of MATLAB, we refer the reader to the MATLAB documentation *Using MATLAB Graphics*. Another excellent resource is the book called *Graphics and GUI's with MATLAB* by Marchand [1999]. These go into more detail on the graphics capabilities in MATLAB that are useful in data analysis such as lighting, use of the camera, animation, etc.

We now describe references that extend the techniques given in this book.

- ***Stem-and-leaf***: Various versions and extensions of the stem-and-leaf plot are available. We show an ordered stem-and-leaf plot in this book, but ordering is not required. Another version shades the leaves. Most introductory applied statistics books have information on stem-and-leaf plots (e.g., Montgomery, et al. [1998]). Hunter [1988] proposes an enhanced stem-and-leaf called the ***digidot plot***. This combines a stem-and-leaf with a time sequence plot. As data are collected they are plotted as a sequence of connected dots and a stem-and-leaf is created at the same time.

- ***Discrete Quantile Plots***: Hoaglin and Tukey [1985] provide similar plots for other discrete distributions. These include the negative binomial, the geometric and the logarithmic series. They also discuss graphical techniques for plotting confidence intervals instead of points. This has the advantage of showing the confidence one has for each count.
- ***Box plots***: Other variations of the box plot have been described in the literature. See McGill, Tukey and Larsen [1978] for a discussion of the variable width box plot. With this type of display, the width of the box represents the number of observations in each sample.
- ***Scatterplots***: Scatterplot techniques are discussed in Carr, et al. [1987]. The methods presented in this paper are especially pertinent to the situation facing analysts today, where the typical data set that must be analyzed is often very large $(n = 10^3, 10^6, \ldots)$. They recommend various forms of binning (including hexagonal binning) and representation of the value by gray scale or symbol area.
- ***PPEDA***: Jones and Sibson [1987] describe a steepest-ascent algorithm that starts from either principal components or random starts. Friedman [1987] combines steepest-ascent with a stepping search to look for a region of interest. Crawford [1991] uses genetic algorithms to optimize the projection index.
- ***Projection Pursuit***: Other uses for projection pursuit have been proposed. These include projection pursuit probability density estimation [Friedman, Stuetzle, and Schroeder, 1984], projection pursuit regression [Friedman and Stuetzle, 1981], robust estimation [Li and Chen, 1985], and projection pursuit for pattern recognition [Flick, et al., 1990]. A 3-D projection pursuit algorithm is given in Nason [1995]. For a theoretical and comprehensive description of projection pursuit, the reader is directed to Huber [1985]. This invited paper with discussion also presents applications of projection pursuit to computer tomography and to the deconvolution of time series. Another paper that provides applications of projection pursuit is Jones and Sibson [1987]. Not surprisingly, projection pursuit has been combined with the grand tour by Cook, et al. [1995]. Montanari and Lizzani [2001] apply projection pursuit to the variable selection problem. Bolton and Krzanowski [1999] describe the connection between projection pursuit and principal component analysis.

Exercises

5.1. Generate a sample of 1000 univariate standard normal random variables using **randn**. Construct a frequency histogram, relative frequency histogram, and density histogram. For the density histogram, superimpose the corresponding theoretical probability density function. How well do they match?

5.2. Repeat problem 5.1 for random samples generated from the exponential, gamma, and beta distributions.

5.3. Do a quantile plot of the Tibetan skull data of Example 5.3 using the standard normal quantiles. Is it reasonable to assume the data follow a normal distribution?

5.4. Try the following MATLAB code using the 3-D multivariate normal as defined in Example 5.18. This will create a slice through the volume at an arbitrary angle. Notice that the colors indicate a normal distribution centered at the origin with the covariance matrix equal to the identity matrix.

```
% Draw a slice at an arbitrary angle
hs = surf(linspace(-3,3,20),...
   linspace(-3,3,20),zeros(20));
% Rotate the surface :
rotate(hs,[1,-1,1],30)
% Get the data that will define the
% surface at an arbitrary angle.
xd = get(hs,'XData');
yd = get(hs,'YData');
zd = get(hs,'ZData');
delete(hs)
% Draw slice:
slice(x,y,z,prob,xd,yd,zd)
axis tight
% Now plot this using the peaks surface as the slice.
% Try plotting against the peaks surface
[xd,yd,zd] = peaks;
slice(x,y,z,prob,xd,yd,zd)
axis tight
```

5.5. Repeat Example 5.23 using the data for *Iris virginica* and *Iris versicolor*. Do the Andrews curves indicate separation between the classes? Do you think it will be difficult to separate these classes based on these features?

5.6. Repeat Example 5.4, where you generate random variables such that

(a) $X \sim N(0, 2)$ and $Y \sim N(0, 1)$

(b) $X \sim N(5, 1)$ and $Y \sim N(0, 1)$

How can you tell from the q-q plot that the scale and the location parameters are different?

5.7. Write a MATLAB program that permutes the axes in a parallel coordinates plot. Apply it to the **iris** data.

5.8. Write a MATLAB program that permutes the order of the variables and plots the resulting Andrews curves. Apply it to the **iris** data.

5.9. Implement Andrews curves using a different set of basis functions as suggested in the text.

5.10. Repeat Example 5.16 and use **rotate3d** (or the rotate toolbar button) to rotate about the axes. Do you see any separation of the different types of insects?

5.11. Do a scatterplot matrix of the *Iris versicolor* data.

5.12. Verify that the two vectors used in Equations 5.24 and 5.25 are orthonormal.

5.13. Write a function that implements Example 5.17 for any data set. The user should have the opportunity to input the labels.

5.14. Define a trivariate normal as your volume, $f(x, y, z)$. Use the MATLAB functions **isosurface** and **isocaps** to obtain contours of constant volume or probability (in this case).

5.15. Construct a quantile plot using the **forearm** data, comparing the sample to the quantiles of a normal distribution. Is it reasonable to model the data using the normal distribution?

5.16. The **moths** data represent the number of moths caught in a trap over 24 consecutive nights [Hand, et al., 1994]. Use the stem-and-leaf to explore the shape of the distribution.

5.17. The **biology** data set contains the number of research papers for 1534 biologists [Tripathi and Gupta, 1988; Hand, et al., 1994]. Construct a binomial plot of these data. Analyze your results.

5.18. In the **counting** data set, we have the number of scintillations in 72 second intervals arising from the radioactive decay of polonium [Rutherford and Geiger, 1910; Hand, et al., 1994]. Construct a Poissonness plot. Does this indicate agreement with the Poisson distribution?

5.19. Use the MATLAB Statistics Toolbox function **boxplot** to compare box plots of the features for each species of **iris** data.

5.20. The **thrombos** data set contains measurements of urinary-thromboglobulin excretion in 12 normal and 12 diabetic patients [van Oost, et al.; 1983; Hand, et al., 1994]. Put each of these into a column of a matrix and use the **boxplot** function to compare normal versus diabetic patients.

5.21. To explore the **shading** options in MATLAB, try the following code from the documentation:

```
% The ezsurf function is available in MATLAB 5.3
% and later.
% First get a surface.
ezsurf('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)',...
    [-6*pi,6*pi])
% Now add some lighting effects:
view(0,75)
shading interp
lightangle(-45,30)
set(findobj('type','surface'),...
    'FaceLighting','phong',...
    'AmbientStrength',0.3,'DiffuseStrength',0.8,...
    'SpecularStrength',0.9,'SpecularExponent',25,...
    'BackFaceLighting','unlit')
axis off
```

5.22. The **bank** data contains two matrices comprised of measurements made on genuine money and forged money. Combine these two matrices into one and use PPEDA to discover any clusters or groups in the data. Compare your results with the known groups in the data.

5.23. Using the data in Example 5.27, do a scatterplot matrix of the original sphered data set. Note the structures in the first four dimensions. Get the first structure and construct another scatterplot matrix of the sphered data after the first structure has been removed. Repeat this process after both structures are removed.

5.24. Load the data sets in **posse**. These contain several data sets from Posse [1995b]. Apply the PPEDA method to these data.