# Chapter 6

## *Finding Structure*

### 6.1 Introduction

In the last chapter, we presented several graphical methods for exploring data sets, but all of them involved static displays. In this chapter, we will be discussing graphical methods for EDA that are dynamic and sometimes allow the user to interact with the display to uncover structure. The power and usefulness of these methods become apparent when used to explore high-dimensional data sets.

Structure can mean many things and is often not defined in advance. In other words, we will know structure when we see it. For example, we might be interested in visually locating groups or possibly finding anomalies (e.g., holes or nonlinear structures or patterns). However, in some of the techniques we will discuss in this chapter, structure means non-Gaussian or some sort of departure from normality.

We will also see that finding structure sometimes involves projecting our data to a lower-dimensional space. Therefore, we start off with a brief introduction to the mechanics of projecting data, along with a discussion of a well-known method for transforming data called principal component analysis. This is followed by a methodology called projection pursuit EDA, where one can find interesting projections of the data, with *interesting* being defined in several ways. The next section has a discussion of independent component analysis and its connection with principal component analysis and projection pursuit. This is followed by dynamic or animated methods that are known as grand tours. We conclude with two techniques for reducing the dimensionality belonging to the family of nonlinear methods: multidimensional scaling and isometric feature mapping.

## 6.2 Projecting Data

The Andrews curves and parallel coordinate plots we presented in the previous chapter allow us 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 dimension via a suitable projection. These methods reduce the data to 1D or 2D 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. This process is called *dimensionality reduction*.

The concept of projecting data is important, and several of the methods described in this chapter project the data to some other space to search for structure or to reduce the dimensionality of the data. For example, principal component analysis, independent component analysis, projection pursuit, and the grand tour all involve projections. So, we will first briefly describe this concept before moving on to specific techniques for finding structure.

### Example 6.1

We illustrate how projection works for a small 2D data set, where we will reduce the dimensionality by projecting to a line. In this case, the projection matrix is given by

$$\mathbf{P} = \begin{bmatrix} (\cos\theta)^2 & \cos\theta\sin\theta \\ \cos\theta\sin\theta & (\sin\theta)^2 \end{bmatrix}.$$

The coordinates of the projected observations are a weighted sum of the original variables, where $\mathbf{P}$ contains the weights. We project the observations using the matrix notation

$$\mathbf{y}_i = \mathbf{P}^T\mathbf{x}_i; \qquad i = 1, \ldots, n. \tag{6.1}$$

The following code shows how to project all of the data at once using the projection matrix.

```
% Specify some data.
X = [-2 4; 2 4;6 1;8 10;7 5;11 8];
% Create the projection matrix.
```

```
theta = pi/4;
c2 = cos(theta)^2;
cs = cos(theta)*sin(theta);
s2 = sin(theta)^2;
P = [c2 cs; cs s2];
% Now project the data onto the line.
Xp = X*P;
% Create the plot shown in Figure 6.1.
plot(Xp(:,1),Xp(:,2),'d',X(:,1),X(:,2),'o')
hold on
plot([0  12 ], [0 12 ], ':')
hold off
axis([-4 12 0 12])
```

The results are shown in Figure 6.1, where one can see that the projected observations now fall on the line. The original data points are shown as circles, and the transformed observations are shown as diamonds.
❑

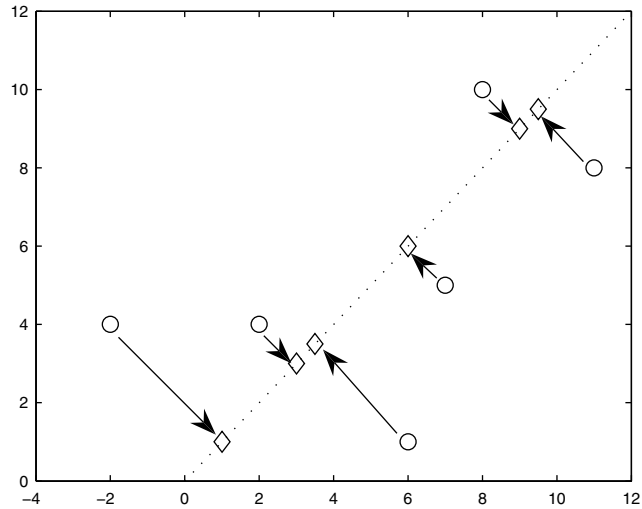


**FIGURE 6.1**
*This illustrates the projection of 2D data (circles) onto a line (diamonds). See Example 6.1 for details.*

The example above did not reduce the dimensionality of the data, since the projected points in the matrix **Xp** are still 2D. We show in the next section how to use a particular projection to obtain data with fewer dimensions. For more information on projections and other matrix transformations, please see Strang [2005] or any linear algebra text.

## 6.3 Principal Component Analysis

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 corresponds to 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. Alternatively, 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. Principal component analysis is discussed in many linear algebra texts [Strang, 2005], and there are also some books dedicated to this subject alone [Jackson, 1991; Joliffe, 2002].

One of the uses of PCA is to map observations to a space such that they are uncorrelated. However, our main interest in the method, as it is used in the search for structure, is to reduce the dimensionality from $d$ to $k$, where $k < d$, and the new set of coordinates are a linear combination of the original variables. We do this by finding $k$ orthogonal linear combinations of the original variables. The first principal component has the largest variance. The second principal component has the next largest variance and is orthogonal to the first one. We can continue in this way until we have the $k$ principal components that explain most of the variance in the original data. The assumption is that the rest of the PCs can be discarded without losing a lot of information.

As noted previously, PCA can start with either the covariance matrix or the correlation matrix. In this text, we will illustrate the procedure using the sample covariance matrix, $\mathbf{S}$. For information on other approaches, such as the decomposition of the correlation matrix and singular value decomposition, see Joliffe [2002] or Jackson [1991].

We showed how to calculate the sample covariance matrix $\mathbf{S}$ in Chapter 3, but we repeat the definition for ease of reading. We first subtract the $d$-dimensional sample mean from each observation to center the data at the sample mean. We denote this centered data matrix as $\mathbf{X}_c$. Note that one should get a vector of zeros (approximately) when using the command **`mean(Xc)`**. The sample covariance matrix is found as

$$\mathbf{S} = \frac{1}{n-1}\mathbf{X}_c^T\mathbf{X}_c,$$

with the *jk*-th element of **S** given by

$$s_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij}-\bar{x}_j)(x_{ik}-\bar{x}_k); \qquad j,k = 1,\ldots,d.$$

The next step is to calculate the eigenvectors and the eigenvalues of the sample covariance matrix. The eigenvalues are found by solving the following equation for each $\lambda_j$, $j = 1,\ldots,d$:

$$|\mathbf{S}-\lambda\mathbf{I}| = 0, \tag{6.2}$$

where **I** is a $d \times d$ identity matrix and $|*|$ denotes the matrix determinant. If one uses the definition of the determinant to expand Equation 6.2, then the result is a polynomial of degree *d*.

Once we have the eigenvalues, we can then find the eigenvectors by solving the following set of equations for vectors $\mathbf{a}_j$:

$$(\mathbf{S}-\lambda_j\mathbf{I})\mathbf{a}_j = 0, \qquad j = 1,\ldots,d,$$

subject to the condition that the set of eigenvectors are orthonormal. This means that the magnitude of each eigenvector is one, and they are orthogonal to each other. Mathematically, this can be expressed as

$$\mathbf{a}_i^T\mathbf{a}_i = 1$$
$$\mathbf{a}_i^T\mathbf{a}_j = 0.$$

We know from matrix algebra that *any* square, symmetric, nonsingular matrix can be transformed to a diagonal matrix using the eigenvectors. In our description here, we use the sample covariance matrix **S**, so the transformation is given by

$$\mathbf{L} = \mathbf{A}^T\mathbf{S}\mathbf{A},$$

where each eigenvector ($\mathbf{a}_j$) of **S** is a column in the matrix **A**, and **L** is a diagonal matrix with the eigenvalues along the diagonal. The eigenvalues are typically ordered as $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$, with the eigenvectors following the same order. This means that the eigenvector $\mathbf{a}_1$ that goes with $\lambda_1$ is the first column of **A**.

The eigenvectors of $\mathbf{S}$ can be used to obtain new transformed variables called ***principal components*** (PCs). The $j$-th PC is given by

$$\mathbf{z}_j = \mathbf{a}_j^T(\mathbf{x} - \bar{\mathbf{x}}), \tag{6.3}$$

for $j = 1, \ldots, d$. We see from Equation 6.3 that the PCs are a linear combination of the original variables where the elements of $\mathbf{a}_j$ provide the coefficients. We transform the data to the principal component coordinate system using the following matrix multiplication:

$$\mathbf{Y} = \mathbf{X}_c\mathbf{A}. \tag{6.4}$$

The matrix $\mathbf{Y}$ in Equation 6.4 contains the ***principal component scores***. The PC scores have zero mean because we first centered the data about the mean, and they are uncorrelated. We could also transform the original observations, but the resulting PC scores would have a different mean. To summarize, the transformed *variables* (Equation 6.3) are the *principal components* and the transformed *observations* (Equation 6.4) are the *PC scores*.

So far, we have not addressed the issue of reducing the dimensionality using PCA, since the PC scores in Equation 6.4 are still $d$-dimensional. We now show how to accomplish this using the eigenvectors that correspond to the $k$ ($k < d$) largest eigenvalues. It can be shown [Joliffe, 2002; Jackson, 1991] that the sum of the eigenvalues is equal to the total variance of the original variables. Therefore, one could include in the projection only those eigenvectors that correspond to the highest eigenvalues. In this way, we are accounting for the largest amount of variation with smaller dimensionality.

We can reduce the dimensionality using the following

$$\mathbf{Y}_k = \mathbf{X}_c\mathbf{A}_k, \tag{6.5}$$

where $\mathbf{A}_k$ contains the first $k$ eigenvectors or columns of $\mathbf{A}$. Note that the matrix $\mathbf{Y}_k$ has $n$ rows and $k$ columns, so each data point is now reduced to $k$ variables.

The issue of what value to use for $k$ is an important one, and several methods have been proposed over the years. We briefly mention just a few of them here and refer the reader to Joliffe [2002], Jackson [1991], and Martinez et al. [2010] for more details. The methods presented here are the scree plot and the cumulative percentage of variance explained.

The ***scree plot*** [Cattell, 1966] is perhaps the most popular method, and as the title suggests, it is graphical rather than quantitative. A scree plot is a plot of the eigenvalues versus the index of the eigenvalue. Depending on the size of the eigenvalues, it might be more informative to plot the log of the eigenvalues. This type of plot is then called a ***log-eigenvalue plot***. To find a value for $k$ using the scree plot, one looks for the elbow in the curve. This is

the part of the curve where it levels off and becomes almost flat. One could also calculate the change in the slope of the lines connecting the points. When the slopes become less steep, then this is a reasonable value for $k$. We will illustrate a scree plot in Example 6.2.

A popular method for determining the number of dimensions $k$ is the ***cumulative percentage of variance explained***. In this method, one selects the $k$ PCs that contribute a cumulative percentage of total variation in the data. This is calculated using

$$t_k = 100 \times \left( \sum_{j=1}^{k} \lambda_j \right) \div \left( \sum_{j=1}^{d} \lambda_j \right).$$

The user must choose a value for $t_d$, but typical values range between 70% and 95%.

### Example 6.2

We illustrate the procedure for PCA using the glass identification data set, downloaded from the UCI Machine Learning Repository (Newman, et al., 1998]. These data were originally used in criminal investigations, where the goal was to classify glass for use as evidence. The data set consists of 214 observations and 9 attributes. We will use the scree plot and the cumulative percentage of variance explained to choose the number of dimensions to keep. First, we load the data, center it, and find the covariance matrix.

```
load glassdata
[n,d] = size(glassdata);
% Center the data.
Xc = glassdata - repmat(sum(glassdata)/n,n,1);
% Find the covariance matrix.
covm = cov(Xc);
```

Next, we use the **eig** function that is part of the main MATLAB® package. The Statistics Toolbox has more extensive functionality for PCA via the **princomp** function.

```
[v,d]= eig(covm);
% Get the eigenvalues and make sure they
% are ordered.
% Also get the indexes of the sort,
% to re-order the columns of the eigenvector matrix.
[eigvals,inds] = sort(diag(d),'descend');
```

We construct the scree plot using the following code, and show the results in Figure 6.2.

```
% Create the scree plot.
plot(1:9, eigvals ,'ko-')
title('Scree Plot - Glass Data')
xlabel('Index')
ylabel('Eigenvalue')
```

We see an elbow at around $k = 3$. The following MATLAB commands will calculate the cumulative percentage of variance.

```
% Now calculate the cumulative percentage of variance.
pervar = 100*cumsum(eigvals)/sum(eigvals);
```

The results are shown here:

```
  47.6205   73.9398   84.7198   94.9223   98.2290
```

We see that approximately 95% of the variance is explained by the first four eigenvalues, indicating that $k = 4$ is a reasonable estimate using this method. Using the estimated number of dimensions of $k = 3$ from the scree plot method, we can project the data to the first three PCs as follows.

```
eigvecs = v(:,inds);
P3 = eigvecs(:,1:3);
Xp3 = Xc*P3;
```

The reader will be asked to explore the principal component scores in the exercises.

❑

## 6.4 Projection Pursuit EDA

There are an infinite number of planes 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 2D projections. However, this might not be the best plane for highlighting interesting and informative structure in the data. Here, ***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 2D view of our data such that the structure (or departure from normality) is maximized over all possible 2D projections.

We can use the Central Limit Theorem to explain why we are interested in departures from normality. Linear combinations of data, including non-normal data, look normal. So, 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 nonnormal projections.

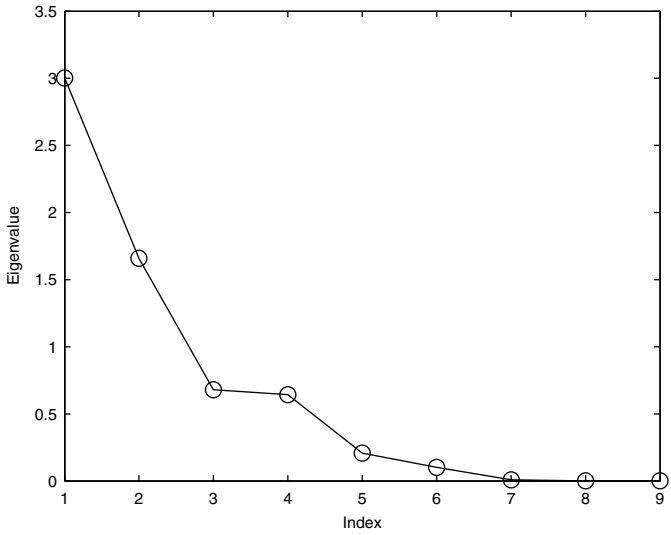


**FIGURE 6.2**
*Here is the scree plot for the glass data. There is an obvious elbow in the curve at three, indicating we should keep three principal components.*

Friedman and Tukey [1974] describe projection pursuit as a way of searching for and exploring nonlinear structure in multi-dimensional data by examining many 2D projections of the data. The idea is that 2D orthogonal projections of the data should reveal structure that is in the original data. The projection pursuit technique can also be used to obtain 1D projections; here we look only at the 2D 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 nonnormality, 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 B, 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 Friedman [1987] to get a sequence of interesting 2D 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*

- $\mathbf{Z}$ is the sphered version of the data matrix $\mathbf{X}$.
- $\mathbf{S}$ is the sample covariance matrix.
- $\alpha, \beta$ 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 $\alpha$ and $\beta$.
- $(z_i^{\alpha}, z_i^{\beta})$ are the sphered observations projected onto the vectors $\alpha$ and $\beta$:

$$z_i^{\alpha} = z_i^T\alpha$$
$$z_i^{\beta} = z_i^T\beta$$

- $(\alpha^*, \beta^*)$ 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 $\alpha$ and $\beta$.
- $\phi_2$ is the standard bivariate normal density.
- $B_k$ is a box in the projection plane.
- $c_k$ is the probability evaluated over the $k$-th region $B_k$ using the standard bivariate normal,

$$c_k = \int\int_{B_k} \phi_2 dz_1 dz_2 .$$

- $I_{B_k}$ 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

$$\alpha(\eta_j) = \alpha\cos\eta_j - \beta\sin\eta_j$$
$$\beta(\eta_j) = \alpha\sin\eta_j + \beta\cos\eta_j$$

- $c$ is a scalar that determines the size of the neighborhood around $(\alpha^*, \beta^*)$ that is visited in the search for planes that provide better values for the projection pursuit index.
- $\mathbf{v}$ 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

The ***Posse chi-square index*** is developed by first dividing the plane into 48 regions or boxes $B_k$ distributed in rings [1995a, 1995b]. See Figure 6.3 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.

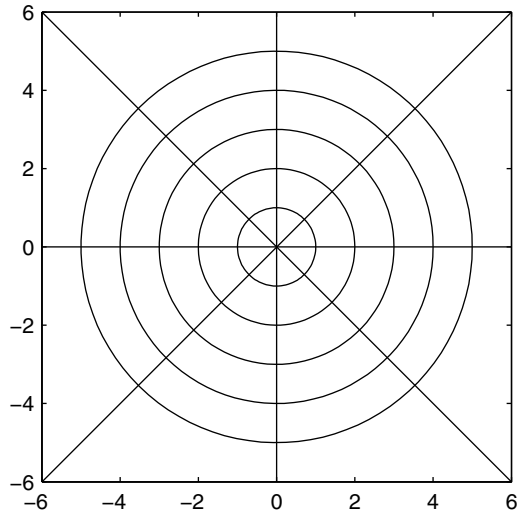


**FIGURE 6.3**
*This shows the layout of the regions $B_k$ for the chi-square projection index [Posse, 1995a].*

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

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

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 B). 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 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 2D 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 $(\alpha^*, \beta^*)$. The method seeks to improve the current best solution by considering two candidate solutions within its neighborhood. These candidate planes are given by

$$\begin{aligned} \mathbf{a}_1 &= \frac{\alpha^* + c\mathbf{v}}{\left\|\alpha^* + c\mathbf{v}\right\|} \qquad \mathbf{b}_1 = \frac{\beta^* - (\mathbf{a}_1^T\beta^*)\mathbf{a}_1}{\left\|\beta^* - (\mathbf{a}_1^T\beta^*)\mathbf{a}_1\right\|} \\ \mathbf{a}_2 &= \frac{\alpha^* - c\mathbf{v}}{\left\|\alpha^* - c\mathbf{v}\right\|} \qquad \mathbf{b}_2 = \frac{\beta^* - (\mathbf{a}_2^T\beta^*)\mathbf{a}_2}{\left\|\beta^* - (\mathbf{a}_2^T\beta^*)\mathbf{a}_2\right\|}. \end{aligned} \tag{6.6}$$

In this approach, we start a global search by looking in large neighborhoods of the current best solution plane $(\alpha^*, \beta^*)$ 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 6.3. 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 $(\alpha^*, \beta^*)$ 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:

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

   where the columns of $\mathbf{Q}$ are the eigenvectors obtained from $\mathbf{S}$, $\Lambda$ is a diagonal matrix of corresponding eigenvalues, and $\mathbf{X}_i$ is the $i$-th observation.
2. Generate a random starting plane, $(\alpha_0, \beta_0)$. This is the current best plane, $(\alpha^*, \beta^*)$.
3. Evaluate the projection index $PI_{\chi^2}(\alpha_0, \beta_0)$ for the starting plane.
4. Generate two candidate planes $(\mathbf{a}_1, \mathbf{b}_1)$ and $(\mathbf{a}_2, \mathbf{b}_2)$ according to Equation 6.6.
5. Find the value of the projection index for these planes, $PI_{\chi^2}(\mathbf{a}_1, \mathbf{b}_1)$ and $PI_{\chi^2}(\mathbf{a}_2, \mathbf{b}_2)$.
6. If one of the candidate planes yields a higher value of the projection pursuit index, then that one becomes the current best plane $(\alpha^*, \beta^*)$.
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.

It is important to 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, 2005] 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

$$\mathbf{T} = \mathbf{U}\mathbf{Z}^T. \tag{6.7}$$

In Equation 6.7, $\mathbf{T}$ 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 $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$.

We now remove the structure that is represented by the first two dimensions. We let $\Theta$ 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:

$$\begin{aligned} \Theta(\mathbf{T}_1) &= \Phi^{-1}[F(\mathbf{T}_1)] \\ \Theta(\mathbf{T}_2) &= \Phi^{-1}[F(\mathbf{T}_2)] \\ \Theta(\mathbf{T}_i) &= \mathbf{T}_i; \qquad i = 3, \ldots, d \ , \end{aligned} \tag{6.8}$$

where $\Phi^{-1}$ is the inverse of the standard normal cumulative distribution function, and $F$ is a function defined next (see Equations 6.9 and 6.10). We see from Equation 6.8 that we will be changing only the first two rows of $\mathbf{T}$.

We now describe the transformation of Equation 6.8 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 $(\alpha^*, \beta^*)$.

Next, we define a rotation about the origin through the angle $\gamma$ as follows

$$\begin{aligned} \tilde{z}_j^{1(t)} &= z_j^{1(t)} \cos\gamma + z_j^{2(t)} \sin\gamma \\ \tilde{z}_j^{2(t)} &= z_j^{2(t)} \cos\gamma - z_j^{1(t)} \sin\gamma, \end{aligned} \tag{6.9}$$

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:

$$z_j^{1(t+1)} = \Phi^{-1}\left\{\frac{r(\tilde{z}_j^{1(t)}) - 0.5}{n}\right\} \qquad z_j^{2(t+1)} = \Phi^{-1}\left\{\frac{r(\tilde{z}_j^{2(t)}) - 0.5}{n}\right\}, \tag{6.10}$$

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 next 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

$$\mathbf{Z}' = \mathbf{U}^T \Theta(\mathbf{U}\mathbf{Z}^T). \tag{6.11}$$

In other words, we transform back using the transpose of the orthonormal matrix $\mathbf{U}$. From matrix theory [Strang, 2005], 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 spherized and then transformed back.

*PROCEDURE – STRUCTURE REMOVAL*

1. Create the orthonormal matrix **U**, where the first two rows of **U** contain the vectors $\alpha^*, \beta^*$.
2. Transform the data **Z** using Equation 6.7 to get **T**.
3. Using only the first two rows of **T**, rotate the observations using Equation 6.9.
4. Normalize each rotated point according to Equation 6.10.
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 $z_j^{1(t+1)}$ and $z_j^{2(t+1)}$, after going through an entire cycle of rotation (Equation 6.9) and normalization (Equation 6.10).
7. Repeat steps 3 through 6 until the projection pursuit index stops changing.
8. Transform the data back using Equation 6.11.

### Example 6.3

We use a synthetic data set to illustrate the MATLAB functions used for PPEDA. 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 - clusters - in two planes.
% Note that the data is in
% ppdata.mat
load ppdata
% For m random starts, find the best projection plane
% using N structure removal procedures.
% Find 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:
[n,d] = size(X);
astar = zeros(d,N);
bstar = zeros(d,N);
ppmax = zeros(1,N);
```

Next we have to sphere the data.

```
% Sphere the data.
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. We can find multiple interesting projections by removing the structure at each iteration of the loop using the function **csppstrtrem**.

```
% 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 6.4 and Figure 6.5, 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. ❑

## 6.5 Independent Component Analysis

***Independent component analysis*** (ICA) is another tool that can be used to help extract information from our data sets. In particular, ICA belongs to a class of techniques called ***blind source separation methods***, where one would like to separate an observed mixture of signals into source signals. For example, one might have two radio stations at different locations, with both of them broadcasting on the same frequency. The mixed signals are picked up by two radio receivers at different geographical locations, and ICA can be used to recover the original source signals.

ICA can be applied to different data such as images, communications channels, stock prices, and others. The method is appropriate when the underlying phenomena or source signals are contained in an observed mixture of noisy signals. As we will see, we do not have to make many assumptions about the source signals to perform ICA.

We will use the example given above of two radio signals from different physical sources to explain ICA. If the two signals are broadcast by two different stations and using a fine time scale, then we can assume that the amplitude of one signal at a particular point in time is unrelated to the amplitude of the other signal at the same point. So, to separate the signals from the mixture, we might look for time-varying signals that are unrelated. The final assumption being that if we find such signals, then they should arise from different physical processes (or broadcasts in our case). Stone [2004] points out that this last assumption going in the reverse direction is not necessarily correct, but it does work in most applications.

Two signals being unrelated can be expressed in terms of statistical independence. As we know from the concepts in Chapter 2, if signals are statistically independent, then the value of one of the signals would not give us any information about the value of other source signals in the mixture. So, ICA seeks to separate the data into a set of statistically independent (or unrelated) component signals, which are then assumed to be from some meaningful source.

We illustrate this idea in Figure 6.6 [Stone, 2004], where we show two signals that are included in the main MATLAB package: a gong and a chirp. The top plot in Figure 6.6 shows the signals over time. If we plot the amplitude of the gong against the amplitude of the chirp (the bottom graphic in Figure 6.6), then we see that there is no obvious pattern or dependency between the two signals.

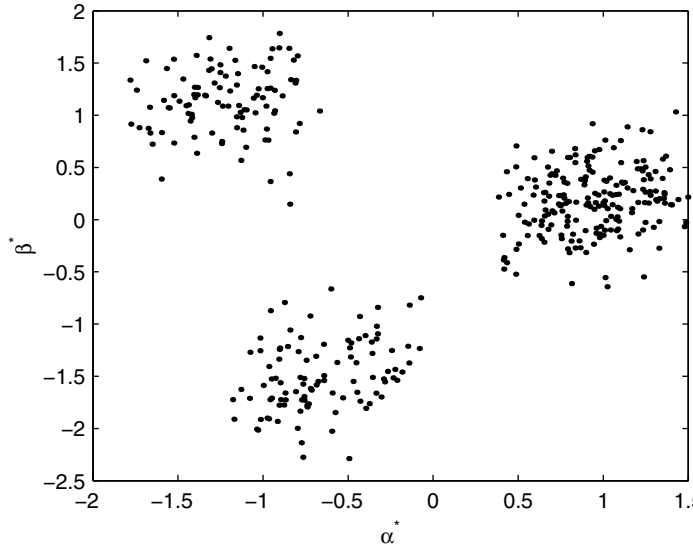


**FIGURE 6.4**
*Here we see the first structure that was found using PPEDA. This structure yields a value of 2.67 for the chi-square projection pursuit index.*

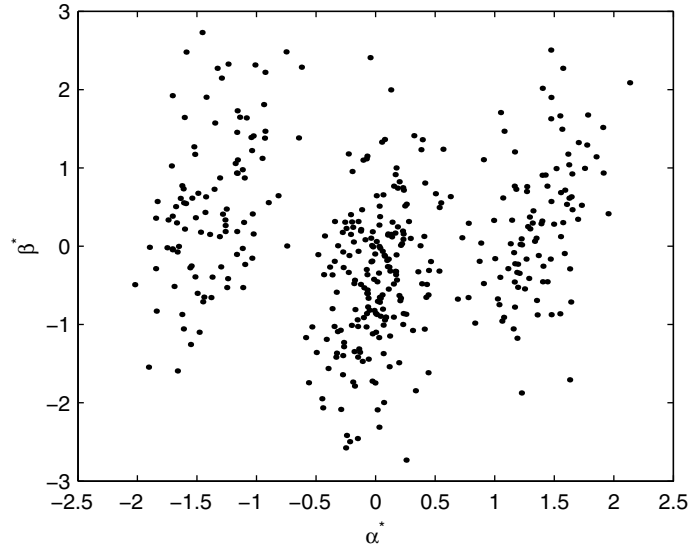


**FIGURE 6.5**
*Here is the second structure we found using PPEDA. This structure has a value of 0.572 for the chi-square projection pursuit index.*

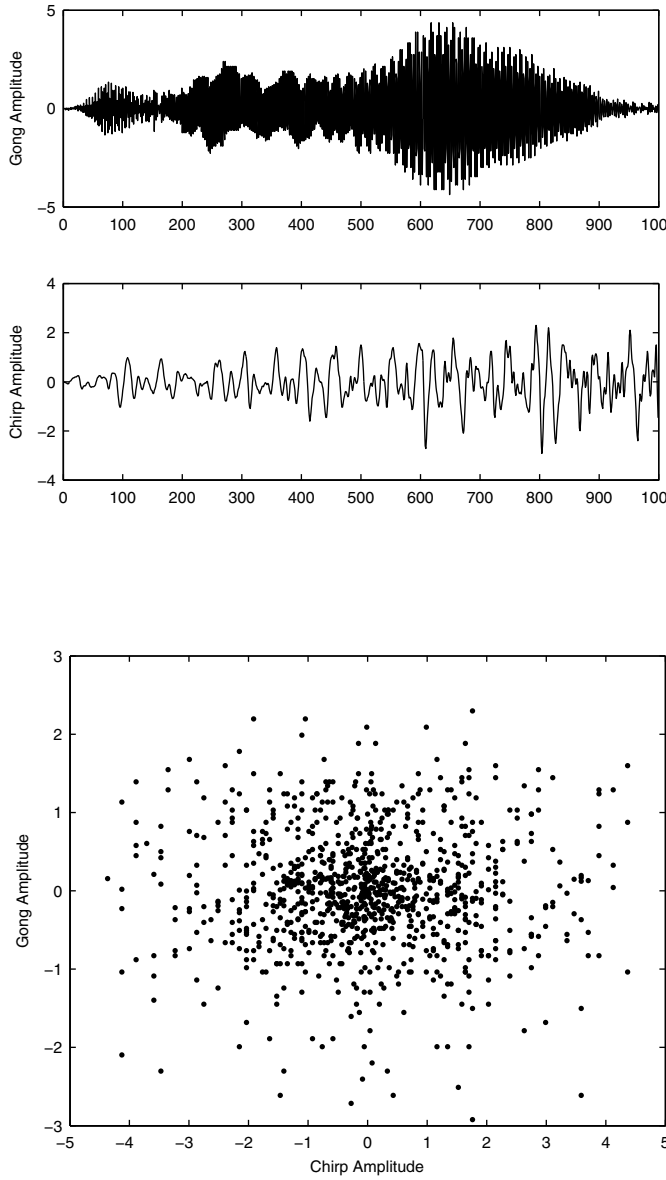


**FIGURE 6.6**
*The top graphic shows the amplitudes of two signals over time. We then plot the amplitude of one signal against the amplitude of the other signal to graphically illustrate the concept of independence between the signals. The result is a cloud of points with no obvious pattern or dependency between the signals [Stone, 2004].*

Note that PCA and ICA are somewhat related in that they are methods that can be used to transform the data. However, a PCA mapping produces *uncorrelated* data (or signals), while ICA finds *nonnormal* and *independent* source signals. Because this is an important issue, we take a moment to discuss the difference between statistical independence and the correlation of random variables [Hyvärinen, 1999].

Without loss of generality, we will illustrate these concepts using only two random variables $X$ and $Y$, both with a mean of zero. We know from Chapter 2 that random variables are independent if the joint probability density function can be written as a product of the marginal probability functions:

$$f(X, Y) = f_X(X) \times f_Y(Y) . \tag{6.12}$$

Equation 6.12 implies that

$$\mathrm{E}[X^p, Y^q] = \mathrm{E}[X^p] \times \mathrm{E}[Y^q] , \tag{6.13}$$

for positive integers, $p$ and $q$. If we take $p = q = 1$, then we have the covariance between $X$ and $Y$. Therefore, Equation 6.13 gives

$$\mathrm{E}[X, Y] = \mathrm{E}[X] \times \mathrm{E}[Y] . \tag{6.14}$$

We know that correlation is a normalized version of the covariance, so if $X$ and $Y$ are uncorrelated, then Equation 6.14 is applicable [Stone, 2004].

To summarize, uncorrelated random variables $X$ and $Y$ imply that the first moment of the joint probability density function is equal to the product of the first moments of the marginals. Independence is a much stricter condition, and it depends on *all* moments of the joint probability density function (Equation 6.13). In general, independence always implies uncorrelatedness, but it does not always go the other way. That is, uncorrelatedness does not imply independence. There is one special case, though, where this is true. If the random variables have a joint multivariate normal distribution, then independence and uncorrelatedness are equivalent.

Formal definitions of ICA can be found in the literature, and Hyvärinen [1999] notes that there are several definitions of ICA. We are interested in the general definition because it has been shown that ICA is a special case of projection pursuit when this definition is used. The general definition simply states that ICA consists of finding a linear transformation so that the resulting components are as independent as possible, and we do this by optimizing a function that measures independence.

We already learned that projection pursuit looks for transformations where the projected data are nonnormal (or interesting), according to some index measuring the degree of the departure from normality. In the case of ICA, the interesting structure we are looking for is independent components. We will

use the kurtosis of the probability density function of a signal as the way to measure the degree of independence or nonnormality in projection pursuit.

We now provide a more mathematical representation of the problem. For simplicity, we will focus on the case of two sound signals and two mixtures (i.e., two receivers). A set of two sound signals can be represented in matrix form as

$$\mathbf{G} = \left[\mathbf{g}_1 \;\; \mathbf{g}_2\right],$$

where $\mathbf{g}_i$ is an $n \times 1$ signal, and $n$ represents the number of observations in each signal.

If we have two sound signals that are received by one sensor, then the output is a ***signal mixture***. This mixture is a weighted sum of the source signals, where the relative proportion of each signal in the mixture depends on several factors. So, for receivers $\mathbf{x}_1$ and $\mathbf{x}_2$, we have signal mixtures given by

$$\begin{aligned}\mathbf{x}_1 &= a\,\mathbf{g}_1 + b\,\mathbf{g}_2 \\ \mathbf{x}_2 &= c\,\mathbf{g}_1 + d\,\mathbf{g}_2\end{aligned}$$

The goal of ICA is to unmix these signals and recover the individual source signals:

$$\begin{aligned}\mathbf{g}_1 &= \alpha\,\mathbf{x}_1 + \beta\,\mathbf{x}_2 \\ \mathbf{g}_2 &= \gamma\,\mathbf{x}_1 + \delta\,\mathbf{x}_2\end{aligned}$$

by estimating the unmixing coefficients. That is, we wish to find a vector $\mathbf{w}$ that maximizes nonnormality (i.e., kurtosis) to extract *estimated signals* $\mathbf{y}$ from a mixture $\mathbf{x}$. We illustrate these concepts in the next example.

### Example 6.4

The following example was adapted from the projection pursuit MATLAB code found in Stone [2004], and it uses a gradient ascent method to find the weight vector that maximizes the kurtosis. The idea is that a projection with maximum kurtosis would show nonnormal structure. It should be noted that we are looking for a 1D projection, and only one signal will be found at each application of the procedure. To find more than one source signal, we would need to apply something similar to the structure removal process, as we explain shortly. We already created a mixture of signals based on the **`chirp`** and **`gong`** signals that are included in the basic MATLAB software. The data are saved in the file called **`icaexample.mat`**.

```
% Load the data.
load icaexample
```

This file has several signal arrays, along with the frequency **N**. One is the matrix of signal mixtures **x**, where each column corresponds to amplitudes recorded at a receiver. We show one of the signal mixtures at the top of Figure 6.7. The file also contains the original unmixed signal matrix **G**, where the first column is the **chirp** signal and the second corresponds to the **gong**. Finally, as in projection pursuit, we spherized the data, producing the matrix **z**. See the online version of this example for the MATLAB code that was used to create the mixture of signals and the matrix **z**. The steps to extract a source signal from **z** are shown here:

```
% Initialize weight vector and normalize.
w = randn(1,2);
w = w/norm(w);
% Specify the step size.
step = 0.02;
% Do projection pursuit using gradient ascent.
for iter=1:100
    % Get estimated source signal, y. The matrix z is
% the spherized signal (see the file for details).
    y = z*w';
    % Get estimated kurtosis. Note that
    % we can use a simplified form because
    % the weight vector has unit length.
    K = mean(y.^4)-3;
    % Find gradient of kurtosis.
    Ycube = repmat(y.^3,1,2);
    grad = mean(Ycube.*z);
    % Update w and normalize.
    w = w + step*grad;
    w = w/norm(w);
end
```

The extracted signal is shown at the bottom of Figure 6.7. We can verify that the extracted signal matches the **chirp** by listening to it. This is done using the following commands.

```
% Play back both signals to compare them.
soundsc(G(:,1),N); % This is the original signal.
soundsc(y,N); % This is the extracted signal.
```

❑

We extracted just one signal in the previous example, but we probably want to find more. After all, this is the idea behind blind source separation. We can extract multiple source signals by first removing the current extracted signal from each of the remaining mixtures using Gram-Schmidt

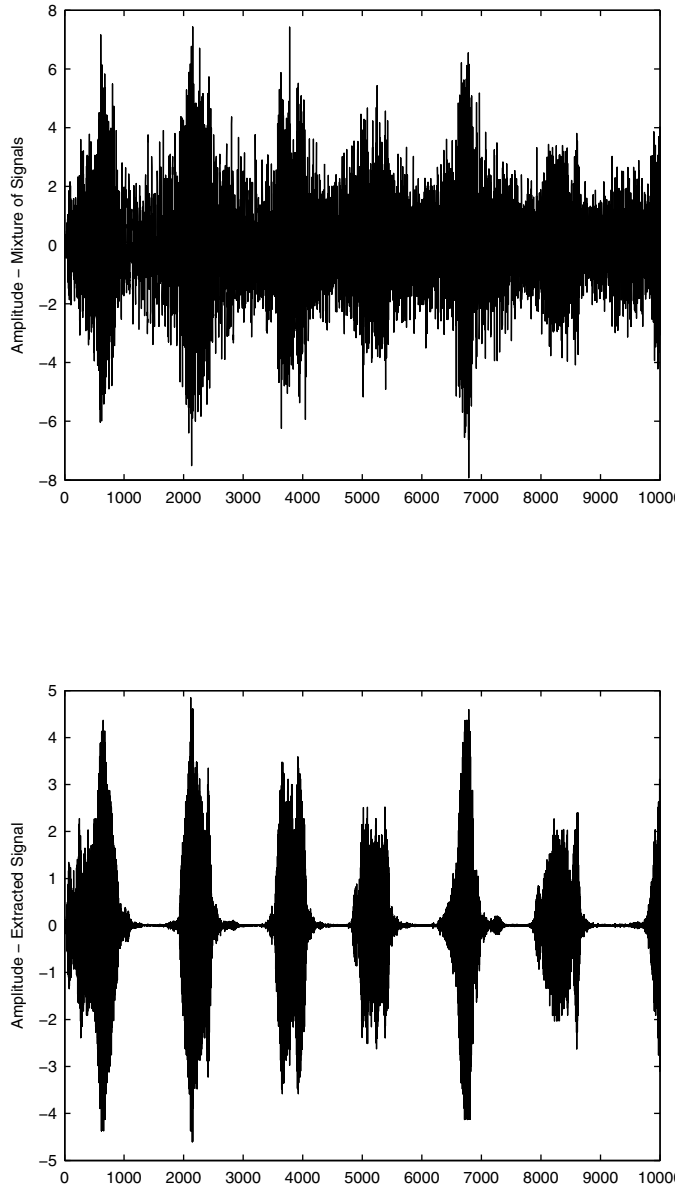


**FIGURE 6.7**
*Th upper plot shows one of the signal mixtures. This is a weighted sum of the* **gong** *and* **chirp** *signals. The lower plot is a picture of the first signal we extracted using projection pursuit with kurtosis measuring the interesting structure.*

orthogonalization [Strang, 2005]. This process is sometimes known as ***deflation***. We then apply the procedure outlined in the previous example to get the next source signal.

We now describe Gram-Schmidt orthogonalization for our two signal, two mixture situation. We denote the original mixtures as $\mathbf{x}_1(0)$ and $\mathbf{x}_2(0)$, and we represent the first extracted signal as $\mathbf{y}_1$. The signal $\mathbf{y}_1$ is removed from the mixtures as follows

$$\mathbf{x}_1(1) = \mathbf{x}_1(0) - \frac{\mathbf{y}_1^T \mathbf{x}_1(0)\mathbf{y}_1}{\mathbf{y}_1^T \mathbf{y}_1}$$

$$\mathbf{x}_2(1) = \mathbf{x}_2(0) - \frac{\mathbf{y}_1^T \mathbf{x}_2(0)\mathbf{y}_1}{\mathbf{y}_1^T \mathbf{y}_1},$$

where $\mathbf{x}_i(1)$ means that one signal has been extracted from the original mixture. This is easily extended to $M$ extracted signals from $M$ signal mixtures. The reader will be asked to implement this in the exercises and to extract the second signal of our mixture.

The independent component analysis methodology has gained widespread use and has become an important tool for analyzing large data sets. Also, there are many versions of ICA, along with various algorithms to implement them. We provide some references for the interested reader at the end of the chapter.

## 6.6 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 onto a plane and then 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 2D 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, which is why we call it 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.

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 1D 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

$$z_i^{\alpha(t)} = \boldsymbol{\alpha}^T(t)\mathbf{x}_i, \tag{6.15}$$

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

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

$$z_i^{\beta(t)} = \beta^T(t)\mathbf{x}_i . \tag{6.16}$$

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

In general, the basic idea of grand tours is to project the data onto a 1D or 2D 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 grand tours are 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 in these tours 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 $\alpha(t)$ and $\beta(t)$. First we consider the data vector $\mathbf{x}$. 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 $\alpha(t)$ to be

$$\alpha(t) = \sqrt{2/d}(\sin\omega_1 t, \cos\omega_1 t, \ldots, \sin\omega_{d/2} t, \cos\omega_{d/2} t), \tag{6.17}$$

and the vector $\beta(t)$ as

$$\beta(t) = \sqrt{2/d}(\cos\omega_1 t, -\sin\omega_1 t, \ldots, \cos\omega_{d/2} t, -\sin\omega_{d/2} t). \tag{6.18}$$

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

The steps for implementing the 2D pseudo grand tour are given here. The details on how to implement this in MATLAB are given in Example 6.5.

*PROCEDURE – PSEUDO GRAND TOUR*

1. Set each $\omega_i$ to an irrational number.
2. Find vectors $\alpha(t)$ and $\beta(t)$ using Equations 6.17 and 6.18.
3. Project the data onto the plane spanned by these vectors using Equations 6.15 and 6.16.
4. Display the projected points, $z_i^{\alpha(t)}$ and $z_i^{\beta(t)}$, in a 2D scatterplot.
5. Using $\Delta t$ 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 6.5

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
```

❑

## 6.7 Nonlinear Dimensionality Reduction

We have already seen several methods for reducing the dimensionality of our data as a way of searching for structure in a data set. These include projection pursuit, the grand tour, principal component analysis, and independent component analysis. While there are many methods for nonlinear dimensionality reduction, we present only two methods. These are multidimensional scaling and isometric feature mapping. Other types of nonlinear dimensionality reduction are locally linear embedding, Hessian eigenmaps, and nonlinear forms of PCA and ICA. We will provide references for these methods at the end of the chapter.

## Multidimensional Scaling

***Multidimensional scaling*** (MDS) encompasses techniques for dimensionality reduction based on proximities or similarities measured on a set of objects. The purpose of MDS is to find a configuration of the data in a low-dimensional space, such that objects that are close together in the full-dimensional space are close together after we reduce the dimensionality. MDS is now widely available in most statistical packages, including the MATLAB Statistics Toolbox.

We will first discuss some notation and definitions before we go on to describe the various categories of MDS [Cox and Cox, 2001]. In general, MDS starts with measures of proximity that quantify how similar or dissimilar observations are to one another. The dissimilarity between observations $\mathbf{x}_r$ and $\mathbf{x}_s$ is denoted by $\delta_{rs}$ and the similarity is denoted by $s_{rs}$. The following is true for *most* definitions of these measures

$$\begin{aligned} \delta_{rs} \geq 0 \qquad & \delta_{rr} = 0 \\ 0 \leq s_{rs} \leq 1 \qquad & s_{rr} = 1 . \end{aligned}$$

This implies that for a dissimilarity measure, small values indicate observations that are close together. The opposite is true for similarity measures; large values indicate observations are similar. These two types of proximity measures can be converted to the other when necessary. Therefore, we can assume the proximities are dissimilarities without loss of generality. We also assume that they are arranged in matrix form, denoted by **D**. The size of this matrix is $n \times n$, and it is a symmetric matrix. Because of its symmetry, it is sometimes given in upper (or lower) triangular form, where only the diagonal and upper (or lower) elements of the matrix are given.

We refer to the distance between observation $r$ and $s$ in the lower-dimensional space by $d_{rs}$. In the MDS literature, the configuration of points in the lower-dimensional space is often denoted by **X**. We follow that convention here, but the reader should be careful not to confuse this with the data matrix. To summarize, with MDS, we start with a dissimilarity matrix **D** and finish with $k$-dimensional transformed observations entered as rows in a matrix **X**.

As with PCA, we must also choose a value for $k$. In MDS, the value for $k$ often depends on the application. If the transformed observations will be used in further analysis, such as classification and clustering, then we might want to use something similar to the scree plot to help us choose $k$. However, when they are used in graphical exploratory data analysis, we might choose $k = 2$ or $k = 3$ for visualizing in scatterplots.

There are many different techniques and algorithms for MDS, most of which fall into two main categories known as metric MDS and nonmetric MDS. The main difference between them comes from the assumptions of

how the dissimilarities $\delta_{rs}$ are mapped to the configuration of distances in the lower dimensional space.

***Metric MDS*** assumes that the dissimilarities $\delta_{rs}$ and distances $d_{rs}$ are related as follows:

$$d_{rs} \approx f(\delta_{rs}), \tag{6.19}$$

where $f$ is a continuous monotonic function. We can see from Equation 6.19, that metric MDS seeks a configuration of points in a lower-dimensional space where distances in that space are approximately the same as some function of the dissimilarities in the full space. Metric MDS is often used in the literature to denote ***classical MDS***, which will be described next. However, metric MDS includes more than this one technique, such as least squares scaling and others [Cox and Cox, 2001].

***Nonmetric MDS*** relaxes the metric properties of Equation 6.19 and requires only that the rank order of the dissimilarities be preserved. Thus, the scaling function must obey the monotonicity constraint:

$$\delta_{rs} < \delta_{ab} \Rightarrow f(\delta_{rs}) \leq f(\delta_{ab}),$$

for all observations. Nonmetric MDS is also known as ***ordinal MDS***.

Most metric and nonmetric MDS solutions require the iterative optimization of an objective function based on the squared discrepancies between $d_{rs}$ and $\delta_{rs}$. However, if the proximity measure in the original $d$-dimensional space and the distance $d_{rs}$ are taken to be Euclidean, then a closed form solution is available to find the configuration of points in a $k$-dimensional space. This is the ***classical MDS*** approach, where the scaling function relating the dissimilarities and distances is the identity function. Thus, we look for a mapping such that

$$d_{rs} = \delta_{rs}.$$

This technique originated with Young and Householder [1938], Torgerson [1952], and Gower [1966]. Gower showed the importance of classical scaling, and he gave it the name ***principal coordinate analysis*** because it has similarities to PCA. We describe the steps of the method next. The reader interested in the derivation can consult Cox and Cox [2001], Borg and Groenen [1997], or Seber [1984].

*PROCEDURE – CLASSICAL MDS*

1. Using the matrix of dissimilarities **D**, find **Q**, where each element of **Q** is given by

$$q_{rs} = -\frac{1}{2}\delta_{rs}^2.$$

2. Find the centered matrix **H** using

$$\mathbf{H} = \mathbf{I} - n^{-1}\mathbf{1}\mathbf{1}^T,$$

where **I** is the $n \times n$ identity matrix, and **1** is a vector of $n$ ones.

3. Find the matrix B, as follows:

$$\mathbf{B} = \mathbf{HQH}.$$

4. Determine the eigenvectors and eigenvalues of **B**.
5. The coordinates in the lower-dimensional space are given by

$$\mathbf{X} = \mathbf{A}_k\mathbf{L}_k^{1/2},$$

where $\mathbf{A}_k$ holds the eigenvectors corresponding to the $k$ largest eigenvalues, and $\mathbf{L}_k^{1/2}$ contains the square root of the $k$ largest eigenvalues along the diagonal.

In some cases, the matrix **B** might have negative eigenvalues. One could ignore the negative eigenvalues and proceed to step 5; other options are given in Cox and Cox [2001]. If the dissimilarities are actually Euclidean distances, then this problem should not come up. It can also be shown that PCA and classical MDS provide equivalent results when the dissimilarities are Euclidean distances.

### Example 6.6

We use the **cereal** data set in this example. Recall that this data set has 8 observations and 11 variables. First we have to load the data and find the interpoint distance (dissimilarity) matrix, **D**. We will use the **pdist** function that is part of the Statistics Toolbox.

```
load cereal
[n,d] = size(cereal);
% Get the matrix of dissimilarities.
D = squareform(pdist(cereal,'cityblock'));
```

The city block metric is given by

$$\delta_{rs} = \sum_{j=1}^{d} |x_{rj} - x_{sj}|.$$

We are now ready to implement the steps for classical MDS, as follows:

```
% Now implement the steps for classical MDS.
Q = -0.5*D.^2;
H = eye(n) - n^(-1)*ones(n,1)*ones(1,n);
B = H*Q*H;
[A,L] = eig(B);
[vals, inds] = sort(diag(L),'descend');
A = A(:,inds);
% Reduce the dimensionality to 2D.
X = A(:,1:2)*diag(sqrt(vals(1:2)));
```

We reduced the data to 2D to make it easier to visualize. We can construct a scatterplot using these steps:

```
% Plot the points and label them.
plot(X(:,1),X(:,2),'o')
text(X(:,1),X(:,2),labs)
title('Classical MDS - City Block Metric')
```

The plot is shown in Figure 6.8. We see from the labels that the sweet cereals are close together, as are some of the non-sweet cereals in the upper right corner. It is interesting to note that corn flakes is off by itself and is not close to any cereal. These results are consistent with the glyph plots we showed in Figure 5.35.

❑

An advantage of using multi-dimensional scaling methods is that one uses the interpoint distance matrix as input to the process rather than the entire data set. If the data set has large $d$ (many variables) and small $n$ (few observations), with $d >> n$, then the interpoint distance matrix is a smaller matrix than the covariance or correlation matrix that would be used in PCA. So, in these cases, MDS is a viable option for dimensionality reduction, when computer memory constraints prohibit the use of something like PCA.

## Isometric Feature Mapping (ISOMAP)

***Isometric feature mapping*** (ISOMAP) belongs to a general class of methods that have come to be known as ***manifold learning***. As we will see shortly, ISOMAP is essentially an application of classical MDS, where estimates of the geodesic distance between points is used to measure the dissimilarity between observations. These manifold learning approaches assume the data lie on a submanifold $M$ of smaller dimension than the full space (see Figure 6.9). The techniques we briefly mention here produce coordinates in a lower dimensional space, such that the neighborhood structure of the submanifold is preserved.

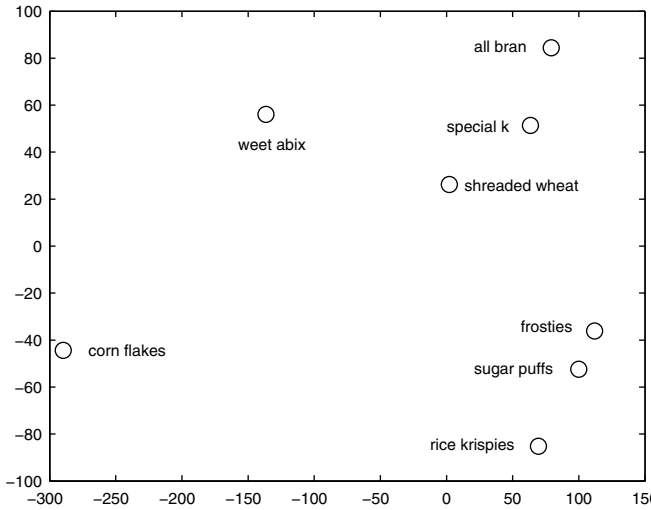


**FIGURE 6.8**
*Here is a scatterplot of the* **cereal** *data where the observations have been reduced to 2D using classical MDS with the city block distance used for the input. Note that similar cereals (e.g., sweet cereals) are close together in this space, indicating that the observations are similar. The reader should compare this with the star and Chernoff face plots (see Chapter 5) of the same data set.*

Several methods for manifold learning have been published in the literature. One of these methods is called ***locally linear embedding*** (LLE) that was developed by Roweis and Saul [2000]. LLE is an eigenvector-based method, and the results do not depend on iteratively solving an optimization problem, as is the case in most versions of MDS and projection pursuit. Another related approach is called ***Hessian eigenmaps*** (HLLE) [Donoho and Grimes, 2003]. An assumption of ISOMAP is that the manifold $M$ is globally isometric to a convex subset of a low-dimensional Euclidean space. Hessian eigenmaps will recover a low-dimensional parametrization for data lying on a manifold that is locally isometric to an open, connected subset of Euclidean space. This expands the class of data sets where isometry principles can be applied to manifold learning.

In our experience, ISOMAP is more stable than these other methods, so we will explain this technique in more detail. ISOMAP was developed by Tenenbaum, de Silva, and Langford [2000] as an application of, or enhancement to, classical MDS. Before explaining ISOMAP in more detail, we first want to illustrate the idea of a submanifold. We use an example found in Tenenbaum, et al. called the Swiss roll. This is illustrated in Figure 6.9. Looking at this picture, we can think of the Swiss roll as a loosely rolled

up (infinitely thin) sheet. Thus, it is really a 2D manifold that is embedded in a 3D space, and the goal of the manifold learning methods is to recover this structure.

The innovation in ISOMAP has to do with the idea that one should use something other than a straight-line Euclidean distance to measure the dissimilarity between points in this type of problem. This concept is illustrated in Figure 6.10, where this time we show a data set that lies along the Swiss roll manifold. We see that points might be close together using Euclidean distance (the two points connected by the line), but they are really far apart when looking at their distance when traveling on the surface of the submanifold. So, Euclidean distance is not a good indication of closeness in terms of the neighborhood structure given by the manifold.

The basic idea, then, of ISOMAP is to use distances along a geodesic path, hopefully measured along the submanifold $M$, as measures of dissimilarity. The ISOMAP method assumes the data lie on an unknown manifold $M$ that is embedded in a high dimensional space. It seeks a mapping that preserves the distances between observations (as in classical MDS), where the distance is given by the geodesic path between points.

The input to the ISOMAP procedure is the interpoint dissimilarity matrix $\mathbf{D}$. First one finds the neighboring points, where the neighborhood is specified by the number of nearest neighbors or a radius $\varepsilon$. The neighborhood relations are then represented as a weighted graph, where the edges of the graph have weights equal to the input distance. The second step of ISOMAP provides estimates of the geodesic distances between all pairs of points $i$ and $j$ by computing their shortest path distance using the neighborhood graph. In the final step, classical MDS is applied to the geodesic distances and an embedding is found in a lower dimensional space. The steps for ISOMAP are outlined next.

*PROCEDURE – ISOMAP*

1. Using the matrix of dissimilarities $\mathbf{D}$, construct the neighborhood graph over all observations by connecting the $ij$-th point if point $i$ and $j$ are in the same neighborhood. Set the lengths of the edges equal to the distance between $i$ and $j$.
2. Calculate the shortest paths between points as given by the graph.
3. Obtain the lower dimensional embedding by applying classical MDS to the geodesic paths found in step 2.

Different embeddings from ISOMAP can be obtained when other neighborhoods or interpoint distances are used as inputs. By changing these inputs, users can explore their data to find different structures. We also note that, depending on the neighborhood and interpoint distance, there might be a zero path length between points. In this case, there will be separate unconnected subsets of points in the lower dimensional embeddings

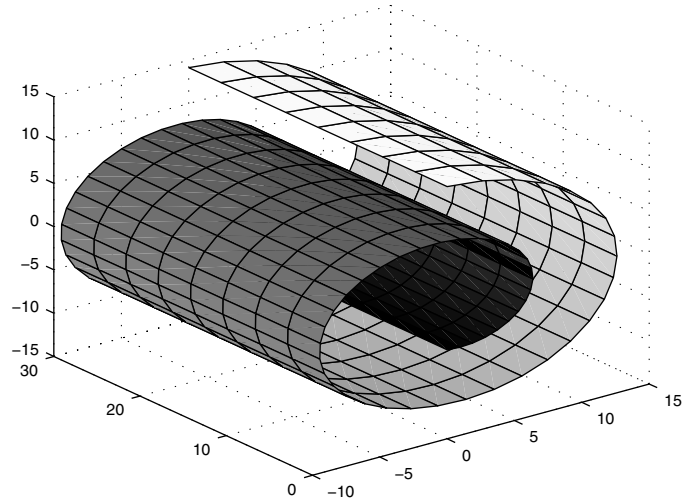


**FIGURE 6.9**
*This shows the submanifold for the Swiss roll. This is a 2D submanifold embedded in a 3D space [Martinez et al., 2010].*

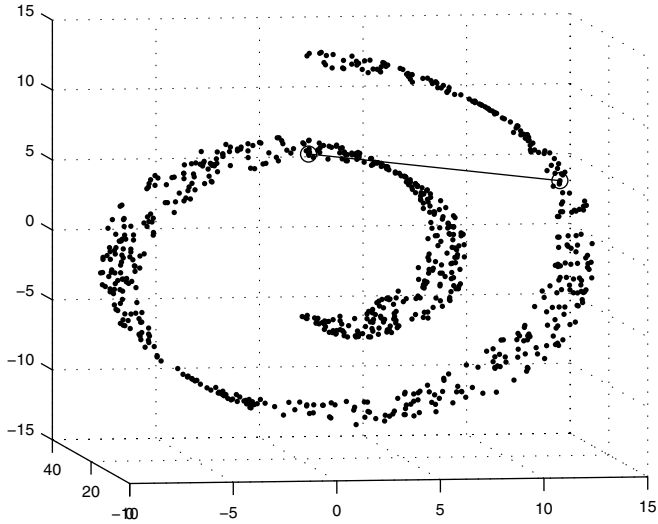


**FIGURE 6.10**
*Here is a data set that was randomly generated according to the Swiss roll parametrization [Tenenbaum, de Silva, and Langford, 2000]. The Euclidean distance between two points is given by the straight line shown here. If we are seeking the neighborhood structure as given by the submanifold M, then it would be better to use the distance between the points if one travels along the manifold [Martinez et al., 2010].*

returned by ISOMAP. If the MATLAB function demonstrated next returns fewer than $n$ points, then one should try increasing the neighborhood size or change the measure of dissimilarity used.

### Example 6.7

We return to the glass data of Example 6.2 to illustrate the use of the ISOMAP MATLAB code provided by Tenenbaum, et al. [2000], which is also included with the Computational Statistics Toolbox. In this example, we show only how to use the ISOMAP function and how to extract the lower dimensional data sets. The input to **isomap** is an $n \times n$ interpoint distance (or dissimilarity) matrix. The MATLAB Statistics Toolbox has two useful functions for calculating dissimilarities: **pdist** and **squareform**. Several dissimilarity metrics are available through **pdist** function, such as Euclidean, standardized Euclidean, Mahalanobis, Minkowski, and others. Some of the metrics are similarity measures that have been converted to dissimilarity (e.g., cosine of the angle between vectors, correlation, Spearman's rank correlation, and Jaccard). The output from **pdist** function is a $1 \times (n \cdot (n-1)/2)$ vector representing the $(n \cdot (n-1)/2)$ pairs of observations. The **isomap** function requires the square interpoint dissimilarity matrix, so we use the **squareform** function to convert the output of **pdist**.

```
% First load the data.
load glassdata
% Then get the interpoint dissimilarity matrix.
% We will use standardized Euclidean distance.
tmp = pdist(glassdata,'seuclidean');
% Now put it into a square matrix.
D = squareform(tmp);
```

The basic syntax for **isomap** is

```
[Y, R, E] = isomap(D, n_fcn, n_size, options);
```

where **D** is the dissimilarity matrix, **n_fcn** specifies the type of neighborhood (**'epsilon'** or **'k'**), and **n_size** is the neighborhood size. See the command line **help** on **isomap** for more information on the **options** argument. The output **Y** is a structure, where the cell array **Y.coords** contains the coordinates for the lower-dimensional embeddings. The default settings have **isomap** returning embedding dimensions of one to ten, so one can get the observations in 1D to 10D spaces. These values are easily changed via the **options** argument.

```
% Now we do ISOMAP. Code is available from
% Tenenbaum, de Silva, and Langford (2000)
% and is included in the Computational Statistics
% Toolbox.We will define the neighborhood using the
% number of nearest neighbors, k = 5.
```

```
[Y,R,E] = isomap(D,'k',5);
```

If default settings are used, then the **isomap** function will produce two plots: a scree plot where the horizontal axis represents the ISOMAP dimensionality (see Figure 6.11) and a scatterplot of the observations in 2D with the neighborhood graph (Figure 6.12). There is a nice elbow in the scree plot for ISOMAP dimensionality of three. The following commands show how to extract the data for a given lower dimensional embedding. It is also important to note that the data sets returned in the cell array **Y.coords** are given with the *n* columns representing the observations, so we have to take the transpose.

```
% We need to extract the data from structure Y.
% Scree plot shows that a value of 3 is a good one.
X3 = Y.coords{3}';
% Plot the data in a scatterplot matrix.
plotmatrix(X3);
```

We display the 3D ISOMAP embedding in Figure 6.13, where we can see some interesting non-Gaussian structures.

❑

## 6.8 MATLAB® Code

There is an open-source EDA Toolbox that has MATLAB functions and GUIs that implement many of the procedures described in this chapter. The EDA Toolbox contains more functions for dimensionality reduction and data tours than what we described here. These include a permutation tour for Andrews' curves and parallel coordinate plots, the pseudo grand tour, the *k*-dimensional grand tour, LLE, HLLE, and others. The toolbox is a companion to the text *Exploratory Data Analysis with MATLAB®, Second Edition* [Martinez, et al., 2010] and can be downloaded from CRC Press.

We include several functions in the Computational Statistics Toolbox that implement some of the algorithms and graphics covered in this chapter. These are summarized in Table 6.1.

Several of the nonlinear dimensionality reduction methods mentioned in this chapter were implemented in MATLAB by the original authors. These include LLE, HLLE, and ISOMAP. Websites for ISOMAP and LLE are

```
http://isomap.stanford.edu/
http://www.cs.toronto.edu/~roweis/lle/
```

There are some useful user-contributed toolboxes for manifold learning and dimensionality reduction available on The MathWorks, Inc. website. Go to the MATLAB Central file exchange at

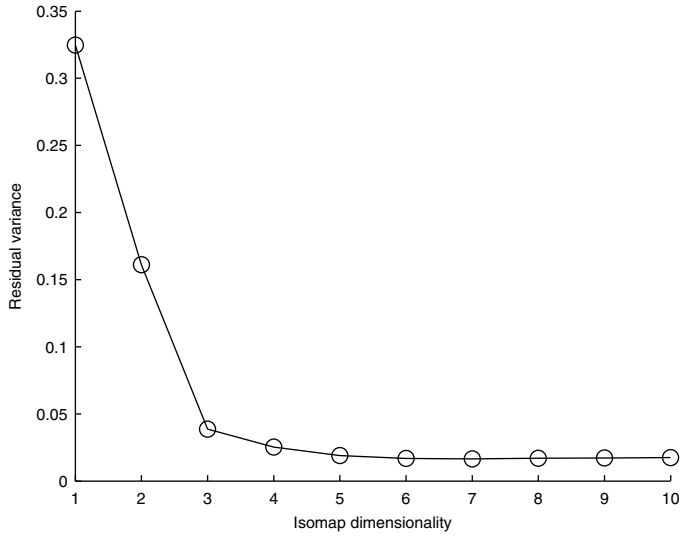


**FIGURE 6.11**
*This shows the scree plot output of* **`isomap`** *of the* **`glassdata`**. *The horizontal axis represents the lower-dimensional embedding size. As with the PCA scree plot, we look for the elbow in the curve. We see from the plot above that 3D seems a reasonable choice for the lower-dimensional embedding.*

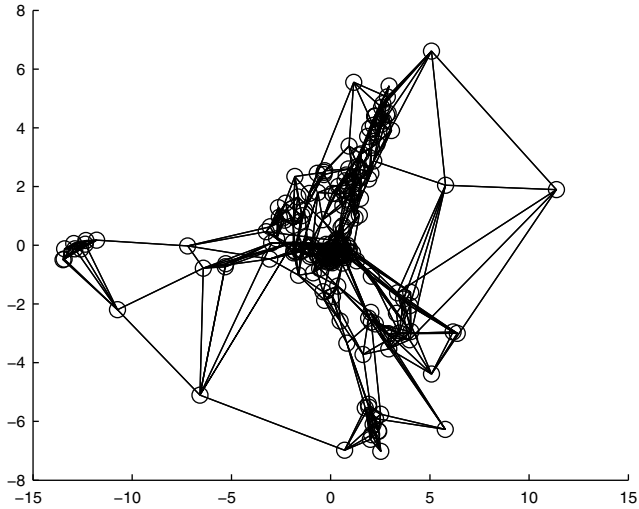


**FIGURE 6.12**
*This is a scatterplot of the 2D ISOMAP coordinates for the* **`glassdata`**. *The neighborhood map is also shown as edges.*

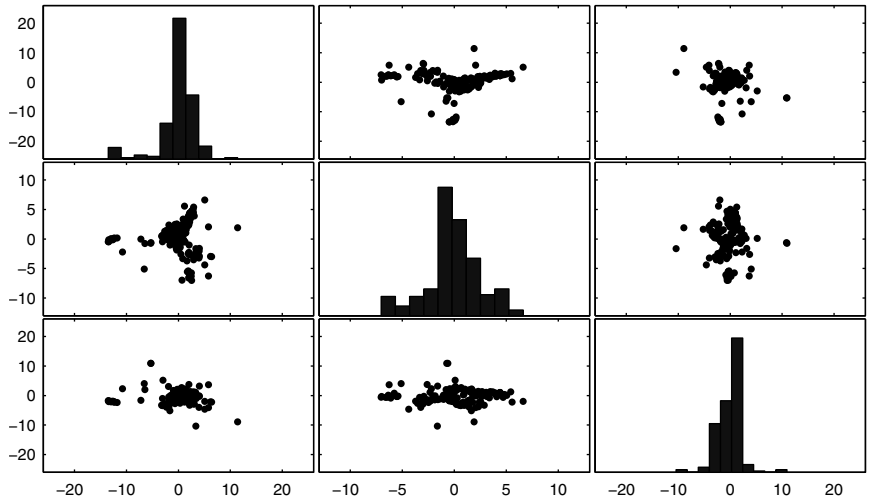


**FIGURE 6.13**
*This scatterplot matrix shows the* **glassdata** *using the 3D ISOMAP coordinates. We see some non-Gaussian structure.*

**http://www.mathworks.com/matlabcentral/**

and search using appropriate keywords.

For more information on independent component analysis, see the following website:

**http://www.cis.hut.fi/projects/ica/**

This website has papers, references, other useful links to ICA, and downloadable MATLAB code. In particular, a wonderful (free) MATLAB toolbox for independent component analysis called FastICA can be downloaded from that website.

The main MATLAB package and the Statistics Toolbox have functions relevant to topics covered in this chapter. These include functions for principal component analysis, classical MDS, metric MDS, and nonmetric MDS. The functions are listed in Table 6.2.

## 6.9 Further Reading

For a general treatment of linear and matrix algebra, we recommend any of the texts by Strang. Some examples are *Linear Algebra and its Applications* [2005] and *Introduction to Linear Algebra* [2003]. These texts also include a discussion of the theory behind principal component analysis, as do most

**TABLE 6.1**

List of Functions from Chapter 6 Included in the Computational Statistics Toolbox

| Purpose | MATLAB Function |
|---|---|
| ISOMAP | `isomap` |
| PPEDA | `csppeda`<br>`csppstrtrem`<br>`csppind` |
| Grand Tour | `torustour` |
| Independent Component Analysis | `csica` |

**TABLE 6.2**

Functions from Base MATLAB and the Statistics Toolbox

| Purpose | MATLAB Function |
|---|---|
| **MATLAB** | |
| Find eigenvectors and eigenvalues | `eig` |
| **Statistics Toolbox** | |
| Principal Component Analysis | `princomp`, `pcacov` |
| Classical MDS | `cmdscale` |
| Nonmetric and metric MDS | `mdscale` |

texts on multivariate data analysis. Some examples of books on multivariate analysis are Manly [2004] and Seber [1984]. As mentioned previously in the chapter, there are also some texts devoted to principal component analysis by Joliffe [2002] and Jackson [1991]. Both of these texts are easy to read and offer many useful examples, in addition to the underlying theory. In our opinion, the Joliffe text tends to be more for the statistician or mathematician, while the Jackson book is more appropriate for engineers.

Multi-dimensional scaling methods are described in most multivariate data analysis texts [Manly 2004]. There are also some books dedicated to this subject. One of these is by Cox and Cox [2001], which is a highly readable text. The other reference is by Borg and Groenen [1997]. Both books describe many of the algorithms used for MDS as a way to aid understanding of the concepts.

We can recommend two books on independent component analysis. First is a book by Stone [2004]. This text is a tutorial introduction to the topic, and it includes many examples and code in MATLAB. Another excellent resource is a book by Hyvärinen, Karhunen, and Oja [2001]. This text is very readable and serves as a good source on theory and applications of ICA. Several

review and tutorial articles have been published on independent component analysis. These include Hyvärinen [1999], Hyvärinen and Oja [2000], and Fodor [2002].

The book by Cook and Swayne [2007] discusses several of the visualization methods discussed in this chapter, such as data tours and multi-dimensional scaling, where they use R and GGobi as their implementation software. For information on R and GGobi, see

**http://www.r-project.org/**

**http://www.ggobi.org/**

For another approach to interactive graphics, see *Visual Statistics* by Young, Valero-Mora, and Friendly [2006]. This book is accessible to those with little statistical training, and it uses a free visual statistics system called ViSta. For information on downloading ViSta, see

**http://www.visualstats.org/**

The paper by Wegman, Carr, and Luo [1993] discusses many of the data tour methods we present in this chapter. This paper and Wegman [1990] also discuss the permutation tour for parallel coordinate plots.

There are several interesting papers on projection pursuit for EDA. 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. Perisic and Posse [2005] propose other indices for projection pursuit EDA that are based on the empirical distribution function and do not have any extra parameters that need to be tuned.

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 3D 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

6.1. Generate some multivariate normal random variables using the following code (or use the corresponding function included in the Computational Statistics Toolbox):

```
mu = zeros(1,2);
sigma = [1 .8; .8 1];
X = mvnrnd(mu, sigma, 100);
```

Verify the covariance by using **corrcoef(X)**. We see that the data points are correlated, as expected. Now, find the eigenvectors of the covariance matrix using

```
eig(cov(X))
```

Use the eigenvectors to project the data as shown in Example 6.1. Verify that the projected observations are uncorrelated.

6.2. Generate 1000, 10D random variables using **randn**. Construct a scree plot and find the cumulative percentage of variance as shown in Example 6.2. Is there any evidence that one could reduce the dimensionality of these data? If so, what would be a value for $k$?

6.3. Looking at Figure 6.2, there appears to be another elbow in the curve at $k = 5$. Reduce the dimensionality of the data to 5D using

```
P5 = eigvecs(:,1,5);
Xp5 = Xc*P5;
```

Do a scatterplot matrix of the 3D and 5D data and compare them. Look at the first observation in 3D and 5D by using **Xp3(1,:)** and **Xp5(1,:)** and compare them.

6.4. 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.

6.5. Using the data in Example 6.3, 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.

6.6. Apply PPEDA to the **glassdata** (Example 6.2). Describe any interesting structure that you find.

6.7. Use Gram-Schmidt orthogonalization to remove the extracted **chirp** signal from the signal mixtures (i.e., the columns of **z**) in Example 6.4. First, listen to the remaining signal mixture. Does it sound like the **gong**? Apply the ICA steps of Example 6.4 to extract the second signal and listen to it using **soundsc**.

6.8. Compare the estimated mixing coefficients of Example 6.4 with the true values in **mixmat**. Do the same for the estimate of the second signal found in the previous problem.

6.9. Apply classical MDS to the **cereal** data using Euclidean distance. Reduce the data to 2D and construct a scatterplot. Compare your results with Figure 6.8.

6.10. Use **gplotmatrix** to do a grouped scatterplot of the ISOMAP results for the **glassdata** (Example 6.7). Does this mapping provide good separation of the groups? Use various graphical techniques (e.g., Andrews curves, parallel coordinate plots, scatterplot matrix, etc.) to explore the other configurations from ISOMAP (i.e., in **Y.coords**).

6.11. Verify that the two vectors used in Equations 6.17 and 6.18 are orthonormal.

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