Chapter 8

Probability Density Estimation

8.1 Introduction

We discussed several techniques for graphical exploratory data analysis in Chapter 5. One purpose of these exploratory techniques is to obtain information and insights about the distribution of the underlying population. For instance, we would like to know if the distribution is multi-modal, skewed, symmetric, etc. Another way to gain understanding about the distribution of the data is to estimate the probability density function from the random sample, possibly using a nonparametric probability density estimation technique.

Estimating probability density functions is required in many areas of computational statistics. One of these is in the modeling and simulation of physical phenomena. We often have measurements from our process, and we would like to use those measurements to determine the probability distribution so we can generate random variables for a Monte Carlo simulation (Chapter 6). Another application where probability density estimation is used is in statistical pattern recognition (Chapter 9). In supervised learning, which is one approach to pattern recognition, we have measurements where each one is labeled with a class membership tag. We could use the measurements for each class to estimate the class-conditional probability density functions, which are then used in a Bayesian classifier. In other applications, we might need to determine the probability that a random variable will fall within some interval, so we would need to evaluate the cumulative distribution function. If we have an estimate of the probability density function, then we can easily estimate the required probability by integrating under the estimated curve. Finally, in Chapter 10, we show how to use density estimation techniques for nonparametric regression.

In this chapter, we cover semi-parametric and nonparametric techniques for probability density estimation. By these, we mean techniques where we make few or no assumptions about what functional form the probability density takes. This is in contrast to a parametric method, where the density is estimated by assuming a distribution and then estimating the parameters.

We present three main methods of semi-parametric and nonparametric density estimation and their variants: histograms, kernel density estimates, and finite mixtures.

In the remainder of this section, we cover some ways to measure the error in functions as background to what follows. Then, in Section 8.2, we present various histogram based methods for probability density estimation. There we cover optimal bin widths for univariate and multivariate histograms, the frequency polygons, and averaged shifted histograms. Section 8.3 contains a discussion of kernel density estimation, both univariate and multivariate. In Section 8.4, we describe methods that model the probability density as a finite (less than n) sum of component densities. As usual, we conclude with descriptions of available MATLAB code and references to the topics covered in the chapter.

Before we can describe the various density estimation methods, we need to provide a little background on measuring the error in functions. We briefly present two ways to measure the error between the true function and the estimate of the function. These are called the mean integrated squared error (MISE) and the mean integrated absolute error (MIAE). Much of the underlying theory for choosing optimal parameters for probability density estimation is based on these concepts.

We start off by describing the mean squared error at a given point in the domain of the function. We can find the ***mean squared error*** **(MSE)** of the estimate $\hat{f}(x)$ at a point x from the following

$$\mathrm{MSE}[\hat{f}(x)] = E[(\hat{f}(x) - f(x))^2]. \tag{8.1}$$

Alternatively, we can determine the error over the domain for x by integrating. This gives us the ***integrated squared error*** **(ISE)**:

$$\mathrm{ISE} = \int (\hat{f}(x) - f(x))^2 dx. \tag{8.2}$$

The ISE is a random variable that depends on the true function $f(x)$, the estimator $\hat{f}(x)$, and the particular random sample that was used to obtain the estimate. Therefore, it makes sense to look at the expected value of the ISE or ***mean integrated squared error***, which is given by

$$\mathrm{MISE} = E\left[\int (\hat{f}(x) - f(x))^2 dx\right]. \tag{8.3}$$

To obtain the ***mean integrated absolute error***, we simply replace the integrand with the absolute difference between the estimate and the true function. Thus, we have

$$\text{MIAE} = E\left[\int \left|\hat{f}(x) - f(x)\right| dx\right]. \tag{8.4}$$

These concepts are easily extended to the multivariate case.

8.2 Histograms

Histograms were introduced in Chapter 5 as a graphical way of summarizing or describing a data set. A histogram visually conveys how a data set is distributed, reveals modes and bumps, and provides information about relative frequencies of observations. Histograms are easy to create and are computationally feasible. Thus, they are well suited for summarizing large data sets. We revisit histograms here and examine optimal bin widths and where to start the bins. We also offer several extensions of the histogram, such as the frequency polygon and the averaged shifted histogram.

1-D Histograms

Most introductory statistics textbooks expose students to the frequency histogram and the relative frequency histogram. The problem with these is that the total area represented by the bins does not sum to 1. Thus, these are not valid probability density estimates. The reader is referred to Chapter 5 for more information on this and an example illustrating the difference between a frequency histogram and a density histogram. Since our goal is to estimate a *bona fide* probability density, we want to have a function $\hat{f}(x)$ that is nonnegative and satisfies the constraint that

$$\int \hat{f}(x)dx = 1 . \tag{8.5}$$

The histogram is calculated using a random sample $X_1, X_2, \ldots, X_n$. The analyst must choose an origin t_0 for the bins and a bin width h. These two parameters define the mesh over which the histogram is constructed. In what follows, we will see that it is the bin width that determines the smoothness of the histogram. Small values of h produce histograms with a lot of variation, while larger bin widths yield smoother histograms. This phenomenon is illustrated in Figure 8.1, where we show histograms with different bin widths. For this reason, the bin width h is sometimes referred to as the ***smoothing parameter***.

Let $B_k = [t_k, t_{k+1})$ denote the k-th bin, where $t_{k+1} - t_k = h$, for all k. We represent the number of observations that fall into the k-th bin by ν_k. The 1-D histogram at a point x is defined as

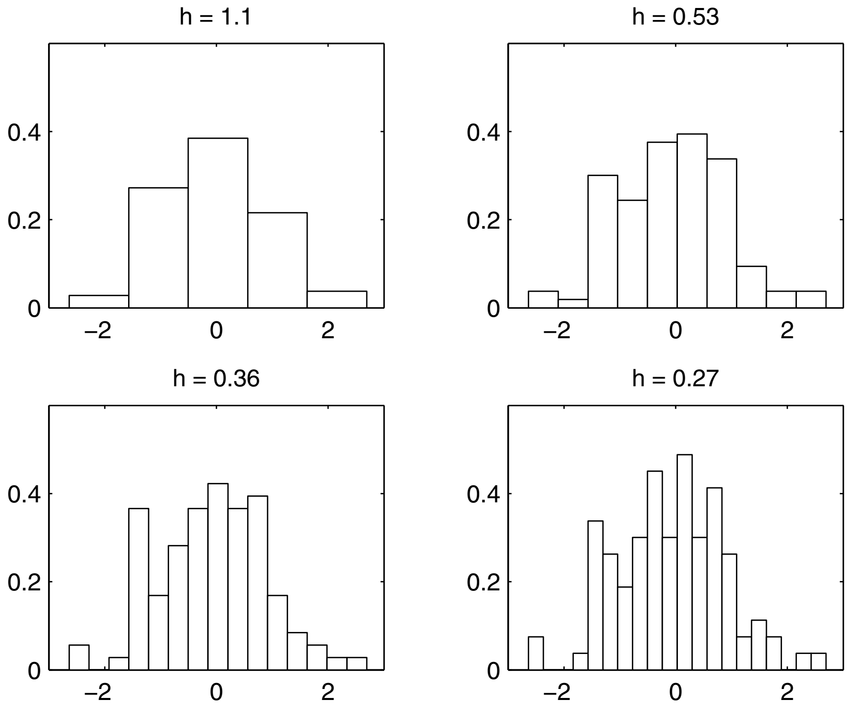

FIGURE 8.1
These are histograms for normally distributed random variables. Notice that for the larger bin widths, we have only one bump as expected. As the smoothing parameter gets smaller, the histogram displays more variation and spurious bumps appear in the histogram estimate.

$$\hat{f}_{Hist}(x) = \frac{v_k}{nh} = \frac{1}{nh}\sum_{i=1}^{n} I_{B_k}(X_i); \qquad x \text{ in } B_k, \tag{8.6}$$

where $I_{B_k}(X_i)$ is the indicator function

$$I_{B_k}(X_i) = \begin{cases} 1, & X_i \text{ in } B_k \\ 0, & X_i \text{ not in } B_k. \end{cases}$$

This means that if we need to estimate the value of the probability density for a given x, then we obtain the value $\hat{f}_{Hist}(x)$ by taking the number of observations in the data set that fall into the same bin as x and multiplying by $1/(nh)$.

Example 8.1
In this example, we illustrate MATLAB code that calculates the estimated value $\hat{f}_{Hist}(x)$ for a given x. We first generate random variables from a standard normal distribution.

```
n = 1000;
x = randn(n,1);
```

We then compute the histogram using MATLAB's **hist** function, using the default value of 10 bins. The issue of the bin width (or alternatively the number of bins) will be addressed shortly.

```
% Get the histogram-default is 10 bins.
[vk,bc] = hist(x);
% Get the bin width.
h = bc(2)- bc(1);
```

We can now obtain our histogram estimate at a point using the following code. Note that we have to adjust the output from **hist** to ensure that our estimate is a *bona fide* density. Let's get the estimate of our function at a point $x_0 = 0$.

```
% Now return an estimate at a point xo.
xo = 0;
% Find all of the bin centers less than xo.
ind = find(bc < xo);
% xo should be between these two bin centers.
b1 = bc(ind(end));
b2 = bc(ind(end)+1);
% Put it in the closer bin.
if (xo-b1) < (b2-xo)   % then put it in the 1st bin
   fhat = vk(ind(end))/(n*h);
else
   fhat = vk(ind(end)+1)/(n*h);
end
```

Our result is **fhat = 0.3477**. The true value for the standard normal evaluated at 0 is $1/\sqrt{2\pi} = 0.3989$, so we see that our estimate is close, but not equal to the true value.
❑

We now look at how we can choose the bin width h. Using some assumptions, Scott [1992] provides the following upper bound for the MSE (Equation 8.1) of $\hat{f}_{Hist}(x)$:

$$\mathrm{MSE}(\hat{f}_{Hist}(x)) \leq \frac{f(\xi_k)}{nh} + \gamma_k^2 h^2; \qquad x \text{ in } B_k, \tag{8.7}$$

where

$$hf(\xi_k) = \int_{B_k} f(t)dt; \qquad \text{for some } \xi_k \text{ in } B_k \ . \tag{8.8}$$

This is based on the assumption that the probability density function $f(x)$ is Lipschitz continuous over the bin interval B_k. A function is ***Lipschitz continuous*** if there is a positive constant γ_k such that

$$|f(x) - f(y)| < \gamma_k |x - y|; \qquad \text{for all } x, y \text{ in } B_k . \tag{8.9}$$

The first term in Equation 8.7 is an upper bound for the variance of the density estimate, and the second term is an upper bound for the squared bias of the density estimate. This upper bound shows what happens to the density estimate when the bin width h is varied.

We can try to minimize the MSE by varying the bin width h. We could set h very small to reduce the bias, but this also increases the variance. The increased variance in our density estimate is evident in Figure 8.1, where we see more spikes as the bin width gets smaller. Equation 8.7 shows a common problem in some density estimation methods: the trade-off between variance and bias as h is changed. Most of the optimal bin widths presented here are obtained by trying to minimize the squared error.

A rule for bin width selection that is often presented in introductory statistics texts is called Sturges' Rule. In reality, it is a rule that provides the number of bins in the histogram, and is given by the following formula.

STURGES' RULE (HISTOGRAM)

$$k = 1 + \log_2 n \ .$$

Here k is the number of bins. The bin width h is obtained by taking the range of the sample data and dividing it into the requisite number of bins, k.

Some improved values for the bin width h can be obtained by assuming the existence of two derivatives of the probability density function $f(x)$. We include the following results (without proof), because they are the basis for many of the univariate bin width rules presented in this chapter. The interested reader is referred to Scott [1992] for more details. Most of what we present here follows his treatment of the subject.

Equation 8.7 provides a measure of the squared error at a point x. If we want to measure the error in our estimate for the entire function, then we can integrate over all values of x. Let's assume $f(x)$ has an absolutely continuous and a square-integrable first derivative. If we let n get very large $(n \to \infty)$, then the asymptotic MISE is

$$\text{AMISE}_{Hist}(h) = \frac{1}{nh} + \frac{1}{12}h^2R(f'), \tag{8.10}$$

where $R(g) \equiv \int g^2(x)dx$ is used as a measure of the roughness of the function, and f' is the first derivative of $f(x)$. The first term of Equation 8.10 indicates the asymptotic integrated variance, and the second term refers to the asymptotic integrated squared bias. These are obtained as approximations to the integrated squared bias and integrated variance [Scott, 1992]. Note, however, that the form of Equation 8.10 is similar to the upper bound for the MSE in Equation 8.7 and indicates the same trade-off between bias and variance, as the smoothing parameter h changes.

The optimal bin width h^*_{Hist} for the histogram is obtained by minimizing the AMISE (Equation 8.10), so it is the h that yields the smallest MISE as n gets large. This is given by

$$h^*_{Hist} = \left(\frac{6}{nR(f')}\right)^{1/3}. \tag{8.11}$$

For the case of data that is normally distributed, we have a roughness of

$$R(f') = \frac{1}{4\sigma^3\sqrt{\pi}}.$$

Using this in Equation 8.11, we obtain the following expression for the optimal bin width for normal data.

NORMAL REFERENCE RULE - 1-D HISTOGRAM

$$h^*_{Hist} = \left(\frac{24\sigma^3\sqrt{\pi}}{n}\right)^{1/3} \approx 3.5\sigma n^{-1/3}. \tag{8.12}$$

Scott [1979, 1992] proposed the sample standard deviation as an estimate of σ in Equation 8.12 to get the following bin width rule.

SCOTT'S RULE

$$\hat{h}^*_{Hist} = 3.5 \times s \times n^{-1/3}.$$

A robust rule was developed by Freedman and Diaconis [1981]. This uses the interquartile range (IQR) instead of the sample standard deviation.

FREEDMAN-DIACONIS RULE

$$\hat{h}^*_{Hist} = 2 \times IQR \times n^{-1/3} .$$

It turns out that when the data are skewed or heavy-tailed, the bin widths are too large using the Normal Reference Rule. Scott [1979, 1992] derived the following correction factor for skewed data:

$$\text{skewness factor}_{Hist} = \frac{2^{1/3}\sigma}{e^{5\sigma^2/4}(\sigma^2+2)^{1/3}(e^{\sigma^2}-1)^{1/2}} . \qquad (8.13)$$

The bin width obtained from Equation 8.12 should be multiplied by this factor when there is evidence that the data come from a skewed distribution. A factor for heavy-tailed distributions can be found in Scott [1992]. If one suspects the data come from a skewed or heavy-tailed distribution, as indicated by calculating the corresponding sample statistics (Chapter 3) or by graphical exploratory data analysis (Chapter 5), then the Normal Reference Rule bin widths should be multiplied by these factors. Scott [1992] shows that the modification to the bin widths is greater for skewness and is not so critical for kurtosis.

Example 8.2

Data representing the waiting times (in minutes) between eruptions of the Old Faithful geyser at Yellowstone National Park were collected [Hand, et al, 1994]. These data are contained in the file **geyser**. In this example, we use an alternative MATLAB function (available in the standard MATLAB package) for finding a histogram, called **histc**. This takes the *bin edges* as one of the arguments. This is in contrast to the **hist** function that takes the *bin centers* as an optional argument. The following MATLAB code will construct a histogram density estimate for the Old Faithful geyser data.

```
load geyser
n = length(geyser);
% Use Normal Reference Rule for bin width.
h = 3.5*std(geyser)*n^(-1/3);
% Get the bin mesh.
t0 = min(geyser)-1;
tm = max(geyser)+1;
rng = tm - t0;
nbin = ceil(rng/h);
bins = t0:h:(nbin*h + t0);
% Get the bin counts vk.
vk = histc(geyser,bins);
% Normalize to make it a bona fide density.
% We do not need the last count in fhat.
fhat(end) = [];
fhat = vk/(n*h);
```

We have to use the following to create a plot of our histogram density. The MATLAB **bar** function takes the bin centers as the argument, so we convert our mesh to bin centers before plotting. The plot is shown in Figure 8.2, and the existence of two modes is apparent.

```
% To plot this, use bar with the bin centers.
tm = max(bins);
bc = (t0+h/2):h:(tm-h/2);
bar(bc,fhat,1,'w')
```

❑

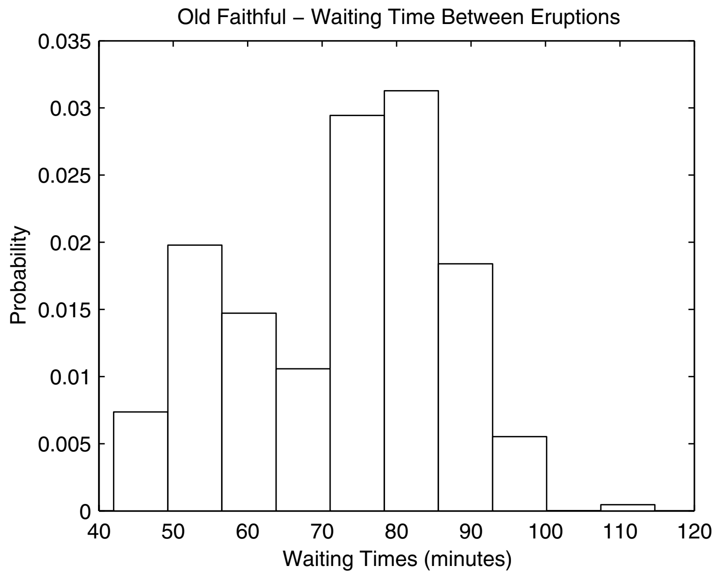

FIGURE 8.2
Histogram of Old Faithful **geyser** data. Here we are using Scott's Rule for the bin widths.

Multivariate Histograms

Given a data set that contains d-dimensional observations $\mathbf{X}_i$, we would like to estimate the probability density $\hat{f}(\mathbf{x})$. We can extend the univariate histogram to d dimensions in a straightforward way. We first partition the d-dimensional space into hyper-rectangles of size $h_1 \times h_2 \times \ldots \times h_d$. We denote the k-th bin by B_k and the number of observations falling into that bin by ν_k, with $\sum \nu_k = n$. The multivariate histogram is then defined as

$$\hat{f}_{Hist}(\mathbf{x}) = \frac{\nu_k}{nh_1h_2\ldots h_d}; \qquad \mathbf{x} \text{ in } B_k. \tag{8.14}$$

If we need an estimate of the probability density at **x**, we first determine the bin that the observation falls into. The estimate of the probability density would be given by the number of observations falling into that same bin divided by the sample size and the bin widths of the partitions. The MATLAB code to create a bivariate histogram was given in Chapter 5. This could be easily extended to the general multivariate case.

For a density function that is sufficiently smooth [Scott, 1992], we can write the asymptotic MISE for a multivariate histogram as

$$\mathrm{AMISE}_{Hist}(\mathbf{h}) = \frac{1}{nh_1h_2\ldots h_d} + \frac{1}{12}\sum_{j=1}^{d} h_j^2 R(f_j), \tag{8.15}$$

where $\mathbf{h} = (h_1, \ldots, h_d)$. As before, the first term indicates the asymptotic integrated variance and the second term provides the asymptotic integrated squared bias. This has the same general form as the 1-D histogram and shows the same bias-variance trade-off. Minimizing Equation 8.15 with respect to h_i provides the following equation for optimal bin widths in the multivariate case

$$h^*_{i_{Hist}} = R(f_i)^{-1/2}\left(6\prod_{j=1}^{d} R(f_j)^{1/2}\right)^{\frac{1}{2+d}} n^{\frac{-1}{2+d}}, \tag{8.16}$$

where

$$R(f_i) = \int_{\mathfrak{R}^d}\left(\frac{\partial}{\partial x_i} f(\mathbf{x})\right)^2 d\mathbf{x}.$$

We can get a multivariate Normal Reference Rule by looking at the special case where the data are distributed as multivariate normal with the covariance equal to a diagonal matrix with $\sigma_1^2, \ldots, \sigma_d^2$ along the diagonal. The Normal Reference Rule in the multivariate case is given below [Scott, 1992].

$$h^*_{i_{Hist}} \approx 3.5\sigma_i n^{\frac{-1}{2+d}}; \qquad i = 1, \ldots, d.$$

Notice that this reduces to the same univariate Normal Reference Rule when $d = 1$. As before, we can use a suitable estimate for σ_i.

Frequency Polygons

Another method for estimating probability density functions is to use a frequency polygon. A univariate frequency polygon approximates the density by linearly interpolating between the bin midpoints of a histogram with equal bin widths. Because of this, the frequency polygon extends beyond the histogram to empty bins at both ends.

The univariate probability density estimate using the frequency polygon is obtained from the following,

$$\hat{f}_{FP}(x) = \left(\frac{1}{2} - \frac{x}{h}\right)\hat{f}_k + \left(\frac{1}{2} + \frac{x}{h}\right)\hat{f}_{k+1}; \qquad \bar{B}_k \le x \le \bar{B}_{k+1}, \tag{8.17}$$

where $\hat{f}_k$ and $\hat{f}_{k+1}$ are adjacent univariate histogram values and $\bar{B}_k$ is the center of bin B_k. An example of a *section* of a frequency polygon is shown in Figure 8.3.

As is the case with the univariate histogram, under certain assumptions, we can write the asymptotic MISE as [Scott, 1992, 1985],

$$\mathrm{AMISE}_{FP}(h) = \frac{2}{3nh} + \frac{49}{2880}h^4 R(f''), \tag{8.18}$$

where f'' is the second derivative of $f(x)$. The optimal bin width that minimizes the AMISE for the frequency polygon is given by

$$h^*_{FP} = 2\left(\frac{15}{49nR(f'')}\right)^{1/5}. \tag{8.19}$$

If $f(x)$ is the probability density function for the standard normal, then $R(f'') = 3/(8\sqrt{\pi}\sigma^5)$. Substituting this in Equation 8.19, we obtain the following Normal Reference Rule for a frequency polygon.

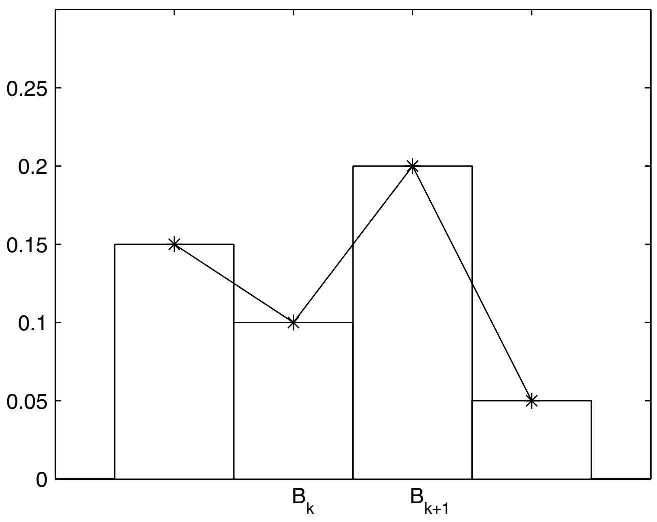

FIGURE 8.3
The frequency polygon is obtained by connecting the center of adjacent bins using straight lines. This figure illustrates a *section* of the frequency polygon.

NORMAL REFERENCE RULE - FREQUENCY POLYGON

$$h^*_{FP} = 2.15\sigma n^{-1/5}.$$

We can use the sample standard deviation in this rule as an estimate of σ or choose a robust estimate based on the interquartile range. If we choose the *IQR* and use $\hat{\sigma} = IQR/1.348$, then we obtain a bin width of

$$\hat{h}^*_{FP} = 1.59 \times IQR \times n^{-1/5}.$$

As for the case of histograms, Scott [1992] provides a skewness factor for frequency polygons, given by

$$\text{skewness factor}_{FP} = \frac{12^{1/5}\sigma}{e^{7\sigma^2/4}(e^{\sigma^2}-1)^{1/2}(9\sigma^4+20\sigma^2+12)^{1/5}}. \qquad (8.20)$$

If there is evidence that the data come from a skewed distribution, then the bin width should be multiplied by this factor. The kurtosis factor for frequency polygons can be found in Scott [1992].

Example 8.3

Here we show how to create a frequency polygon using the Old Faithful **geyser** data. We must first create the histogram from the data, where we use the frequency polygon Normal Reference Rule to choose the smoothing parameter.

```
load geyser
n = length(geyser);
% Use Normal Reference Rule for bin width
% of frequency polygon.
h = 2.15*sqrt(var(geyser))*n^(-1/5);
t0 = min(geyser)-1;
tm = max(geyser)+1;
bins = t0:h:tm;
vk = histc(geyser,bins);
vk(end) = [];
fhat = vk/(n*h);
```

We then use the MATLAB function called **interp1** to interpolate between the bin centers. This function takes three arguments (and an optional fourth argument). The first two arguments to **interp1** are the **xdata** and **ydata** vectors that contain the observed data. In our case, these are the bin centers and the bin heights from the density histogram. The third argument is a vector of **xinterp** values for which we would like to obtain interpolated **yinterp** values. There is an optional fourth argument that allows the user to select the type of interpolation (**linear**, **cubic**, **nearest** and **spline**). The default is **linear**, which is what we need for the frequency polygon. The following code constructs the frequency polygon for the **geyser** data.

```
% For frequency polygon, get the bin centers,
% with empty bin center on each end.
bc2 = (t0-h/2):h:(tm+h/2);
binh = [0 fhat 0];
% Use linear interpolation between bin centers
% Get the interpolated values at x.
xinterp = linspace(min(bc2),max(bc2));
fp = interp1(bc2, binh, xinterp);
```

To see how this looks, we can plot the frequency polygon and underlying histogram, which is shown in Figure 8.4.

```
% To plot this, use bar with the bin centers
tm = max(bins);
bc = (t0+h/2):h:(tm-h/2);
bar(bc,fhat,1,'w')
hold on
plot(xinterp,fp)
hold off
```

```
axis([30 120 0 0.035])
xlabel('Waiting Time (minutes)')
ylabel('Probability Density Function')
title('Old Faithful-Waiting Times Between Eruptions')
```

To ensure that we have a valid probability density function, we can verify that the area under the curve is approximately one by using the `trapz` function.

```
area = trapz(xinterp,fp);
```

We get an approximate area under the curve of 0.9998, indicating that the frequency polygon is indeed a *bona fide* density estimate.
❑

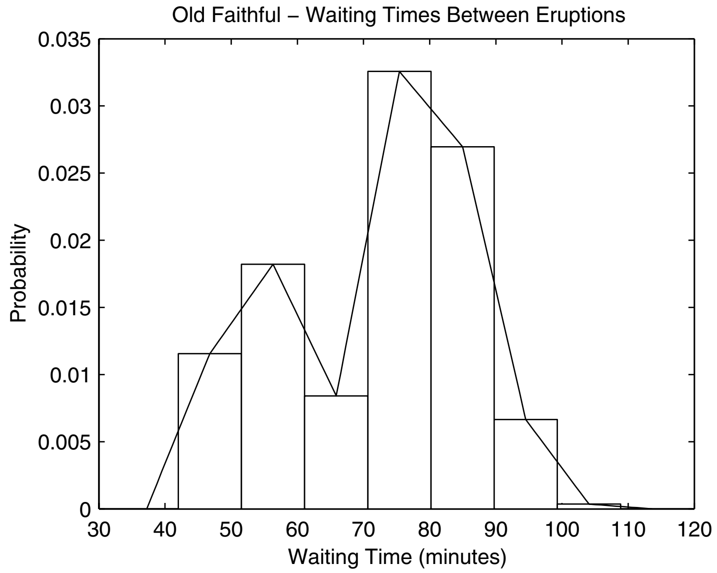

FIGURE 8.4
Frequency polygon for the Old Faithful data.

The frequency polygon can be extended to the multivariate case. The interested reader is referred to Scott [1985, 1992] for more details on the multivariate frequency polygon. He proposes an approximate Normal Reference Rule for the multivariate frequency polygon given by the following formula.

NORMAL REFERENCE RULE - FREQUENCY POLYGON (MULTIVARIATE)

$$h_i^* = 2\sigma_i n^{-1/(4+d)},$$

where a suitable estimate for σ_i can be used. This is derived using the assumption that the true probability density function is multivariate normal with covariance equal to the identity matrix. The following example illustrates the procedure for obtaining a bivariate frequency polygon in MATLAB.

Example 8.4

We first generate some random variables that are bivariate standard normal and then calculate the surface heights corresponding to the linear interpolation between the histogram density bin heights.

```
% First get the constants.
bin0 = [-4 -4];
n = 1000;
% Normal Reference Rule with sigma = 1.
h = 3*n^(-1/4)*ones(1,2);
% Generate bivariate standard normal variables.
x = randn(n,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 or bin edges.
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);
```

Now that we have the random variables and the bin edges, the next step is to find the number of observations that fall into each bin. This is easily done with the MATLAB function **inpolygon**. This function can be used with any polygon (e.g., triangle or hexagon), and it returns the indices to the points that fall into that polygon.

```
% 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
```

```
    fhat = vu/(n*h(1)*h(2));
```

Now that we have the histogram density, we can use the MATLAB function **interp2** to linearly interpolate at points between the bin centers.

```
    % Now get the bin centers for the frequency polygon.
    % We add bins at the edges with zero height.
    t1 = (bin0(1)-h(1)/2):h(1):(max(t1)+h(1)/2);
    t2 = (bin0(2)-h(2)/2):h(2):(max(t2)+h(2)/2);
    [bcx,bcy] = meshgrid(t1,t2);
    [nr,nc] = size(fhat);
    binh = zeros(nr+2,nc+2);  % add zero bin heights
    binh(2:(1+nr),2:(1+nc))=fhat;
    % Get points where we want to interpolate to get
    % the frequency polygon.
    [xint,yint]=meshgrid(linspace(min(t1),max(t1),30),...
       linspace(min(t2),max(t2),30));
    fp = interp2(bcx,bcy,binh,xint,yint,'linear');
```

We can verify that this is a valid density by estimating the area under the curve.

```
    df1 = xint(1,2)-xint(1,1);
    df2 = yint(2,1)-yint(1,1);
    area = sum(sum(fp))*df1*df2;
```

This yields an area of 0.9976. A surface plot of the frequency polygon is shown in Figure 8.5.
❑

Averaged Shifted Histograms

When we create a histogram or a frequency polygon, we need to specify a complete mesh determined by the bin width h and the starting point t_0. The reader should have noticed that the parameter t_0 did not appear in any of the asymptotic integrated squared bias or integrated variance expressions for the histograms or frequency polygons. The MISE is affected more by the choice of bin width than the choice of starting point t_0. The averaged shifted histogram (ASH) was developed to account for different choices of t_0, with the added benefit that it provides a *'smoother'* estimate of the probability density function.

The idea is to create many histograms with different bin origins t_0 (but with the same h) and average the histograms together. The histogram is a piecewise constant function, and the average of piecewise constant functions will also be the same type of function. Therefore, the ASH is also in the form of a histogram, and the following discussion treats it as such. The ASH is often implemented in conjunction with the frequency polygon, where the latter is used to linearly interpolate between the smaller bin widths of the ASH.

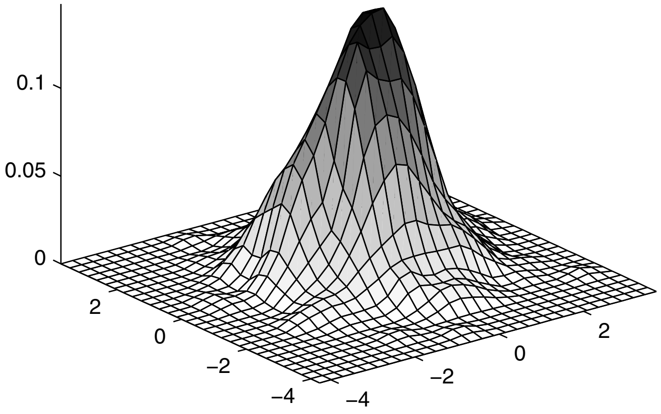

FIGURE 8.5.
Frequency polygon of bivariate standard normal data.

To construct an ASH, we have a set of m histograms, $\hat{f}_1, \ldots, \hat{f}_m$ with constant bin width h. The origins are given by the sequence

$$t'_0 = t_0 + 0, t_0 + \frac{h}{m}, t_0 + \frac{2h}{m}, \ldots, t_0 + \frac{(m-1)h}{m}.$$

In the univariate case, the unweighted or naive ASH is given by

$$\hat{f}_{ASH}(x) = \frac{1}{m}\sum_{i=1}^{m}\hat{f}_i(x), \tag{8.21}$$

which is just the average of the histogram estimates at each point x. It should be clear that the $\hat{f}_{ASH}$ is a piecewise function over smaller bins, whose width is given by $\delta = h/m$. This is shown in Figure 8.6 where we have a single histogram $\hat{f}_i$ and the ASH estimate.

In what follows, we consider the ASH as a histogram over the narrower intervals given by $B'_k = [k\delta, (k+1)\delta)$, with $\delta = h/m$. As before we denote the bin counts for these bins by ν_k. An alternative expression for the naive ASH can be written as

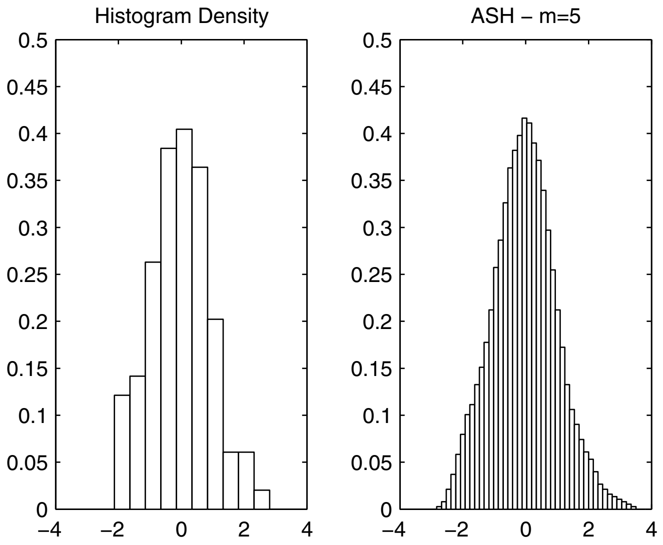

FIGURE 8.6
On the left is a histogram density based on 100 standard normal random variables, where we used the MATLAB default of 10 bins. On the right is an ASH estimate for the same data set, with $m = 5$.

$$\hat{f}_{ASH}(x) = \frac{1}{nh} \sum_{i=1-m}^{m-1} \left(1 - \frac{|i|}{m}\right) \nu_{k+i}; \qquad x \text{ in } B'_k. \tag{8.22}$$

To make this a little clearer, let's look at a simple example of the naive ASH, with $m = 3$. In this case, our estimate at a point x is

$$\hat{f}_{ASH}(x) = \frac{1}{nh}\left[\left(1 - \frac{2}{3}\right)\nu_{k-2} + \left(1 - \frac{1}{3}\right)\nu_{k-1} + \left(1 - \frac{0}{3}\right)\nu_{k-0} + \right.$$
$$\left.\left(1 - \frac{1}{3}\right)\nu_{k+1} + \left(1 - \frac{2}{3}\right)\nu_{k+2}\right]; \qquad x \text{ in } B'_k.$$

We can think of the factor $(1 - |i|/m)$ in Equation 8.22 as weights on the bin counts. We can use arbitrary weights instead, to obtain the general ASH.

GENERAL AVERAGED SHIFTED HISTOGRAM

$$\hat{f}_{ASH} = \frac{1}{nh} \sum_{|i| < m} w_m(i) \nu_{k+i}; \qquad x \text{ in } B'_k\ . \tag{8.23}$$

A general formula for the weights is given by

$$w_m(i) = m \times \frac{K(i/m)}{\sum_{j=1-m}^{m-1} K(j/m)}; \qquad i = 1-m, \ldots, m-1 \ , \tag{8.24}$$

with K a continuous function over the interval $[-1, 1]$. This function K is sometimes chosen to be a probability density function. In Example 8.5, we use the biweight function:

$$K(t) = \frac{15}{16}(1-t^2)^2 I_{[-1,1]}(t) \tag{8.25}$$

for our weights. Here $I_{[-1,1]}$ is the indicator function over the interval $[-1, 1]$.

The algorithm for the general univariate ASH [Scott, 1992] is given here and is also illustrated in MATLAB in Example 8.5. This algorithm requires at least $m-1$ empty bins on either end.

UNIVARIATE ASH - ALGORITHM:

```
1. Generate a mesh over the range $(t_0, nbin \times \delta + t_0)$ with bin widths
   of size $\delta$, $\delta << h$ and $h = m\delta$. The quantity nbin is the number of
   bins - see the comments below for more information on this num-
   ber. Include at least m - 1 empty bins on either end of the range.
2. Compute the bin counts $\nu_k$.
3. Compute the weight vector $w_m(i)$ given in Equation 8.24.
4. Set all $\hat{f}_k = 0$.
5. Loop over $k = 1$ to nbin
   Loop over $i = max\{1, k-m+1\}$ to $min\{nbin, k+m-1\}$

         Calculate: $\hat{f}_i = \hat{f}_i + \nu_k w_m(i-k)$.

6. Divide all $\hat{f}_k$ by nh, these are the ASH heights.
7. Calculate the bin centers using $\bar{B}_k = t_0 + (k-0.5)\delta$.
```

In practice, one usually chooses the *m* and *h* by setting the number of narrow (size δ) bins between 50 and 500 over the range of the sample. This is then extended to put some empty bins on either end of the range.

Example 8.5
In this example, we construct an ASH probability density estimate of the Buffalo **snowfall** data [Scott, 1992]. These data represent the annual snowfall in inches in Buffalo, New York over the years 1910-1972. First load the data and get the appropriate parameters.

```
load snowfall
n = length(snowfall);
m = 30;
h = 14.6;
delta = h/m;
```

The next step is to construct a mesh using the smaller bin widths of size δ over the desired range. Here we start the density estimate at zero.

```
% Get the mesh.
t0 = 0;
tf = max(snowfall)+20;
nbin = ceil((tf-t0)/delta);
binedge = t0:delta:(t0+delta*nbin);
```

We need to obtain the bin counts for these smaller bins, and we use the **histc** function since we want to use the bin edges rather than the bin centers.

```
% Get the bin counts for the smaller binwidth delta.
vk = histc(snowfall,binedge);
% Put into a vector with m-1 zero bins on either end.
fhat = [zeros(1,m-1),vk,zeros(1,m-1)];
```

Next, we construct our weight vector according to Equation 8.24, where we use the biweight kernel given in Equation 8.25. Instead of writing the kernel as a separate function, we will use the MATLAB **inline** function to create a function object. We can then call that **inline** function just as we would an M-file function.

```
% Get the weight vector.
% Create an inline function for the kernel.
kern = inline('(15/16)*(1-x.^2).^2');
ind = (1-m):(m-1);
% Get the denominator.
den = sum(kern(ind/m));
% Create the weight vector.
wm = m*(kern(ind/m))/den;
```

The following section of code essentially implements steps 5 - 7 of the ASH algorithm.

```
% Get the bin heights over smaller bins.
fhatk = zeros(1,nbin);
for k = 1:nbin
```

```
      ind = k:(2*m+k-2);
      fhatk(k) = sum(wm.*fhat(ind));
   end
   fhatk = fhatk/(n*h);
   bc = t0+((1:k)-0.5)*delta;
```

We use the following steps to obtain Figure 8.7, where we use a different type of MATLAB plot to show the ASH estimate. We use the bin edges with the **stairs** plot, so we must append an extra bin height at the end to ensure that the last bin is drawn and to make it dimensionally correct for plotting.

```
   % To use the stairs plot, we need to use the
   % bin edges.
   stairs(binedge,[fhatk fhatk(end)])
   axis square
   title('ASH - Buffalo Snowfall Data')
   xlabel('Snowfall (inches)')
```

❑

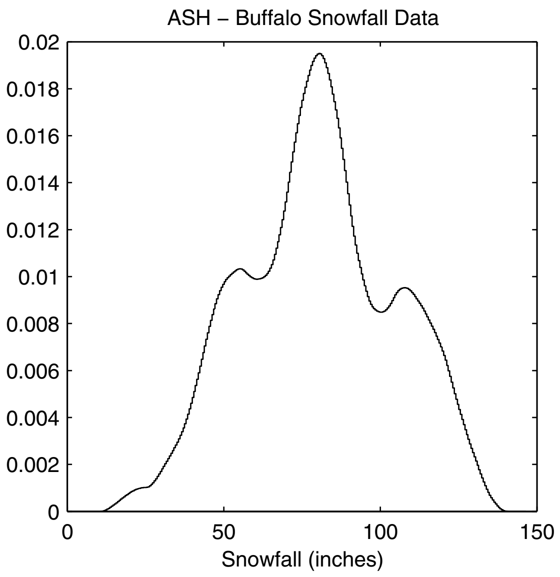

FIGURE 8.7
ASH estimate for the Buffalo snowfall data. The parameters used to obtain this were h = 14.6 inches and m = 30. Notice that the ASH estimate reveals evidence of three modes.

The multivariate ASH is obtained by averaging shifted multivariate histograms. Each histogram has the same bin dimension $h_1 \times \ldots \times h_d$, and each is constructed using shifts along the coordinates given by multiples of δ_i/m_i, $i = 1, \ldots, d$. Scott [1992] provides a detailed algorithm for the bivariate ASH.

8.3 Kernel Density Estimation

Scott [1992] shows that as the number of histograms m approaches infinity, the ASH becomes a kernel estimate of the probability density function. The first published paper describing nonparametric probability density estimation was by Rosenblatt [1956], where he described the general kernel estimator. Many papers that expanded the theory followed soon after. A partial list includes Parzen [1962], Cencov [1962] and Cacoullos [1966]. Several references providing surveys and summaries of nonparametric density estimation are provided in Section 8.7. The following treatment of kernel density estimation follows that of Silverman [1986] and Scott [1992].

Univariate Kernel Estimators

The kernel estimator is given by

$$\hat{f}_{Ker}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right), \tag{8.26}$$

where the function $K(t)$ is called a ***kernel***. This must satisfy the condition that $\int K(t)dt = 1$ to ensure that our estimate in Equation 8.26 is a *bona fide* density estimate. If we define $K_h(t) = K(t/h)/h$, then we can also write the kernel estimate as

$$\hat{f}_{Ker}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - X_i)\,. \tag{8.27}$$

Usually, the kernel is a symmetric probability density function, and often a standard normal density is used. However, this does not have to be the case, and we will present other choices later in this chapter. From the definition of a kernel density estimate, we see that our estimate $\hat{f}_{Ker}(x)$ inherits all of the properties of the kernel function, such as continuity and differentiability..

From Equation 8.26, the estimated probability density function is obtained by placing a weighted kernel function, centered at each data point and then taking the average of them. See Figure 8.8 for an illustration of this procedure.

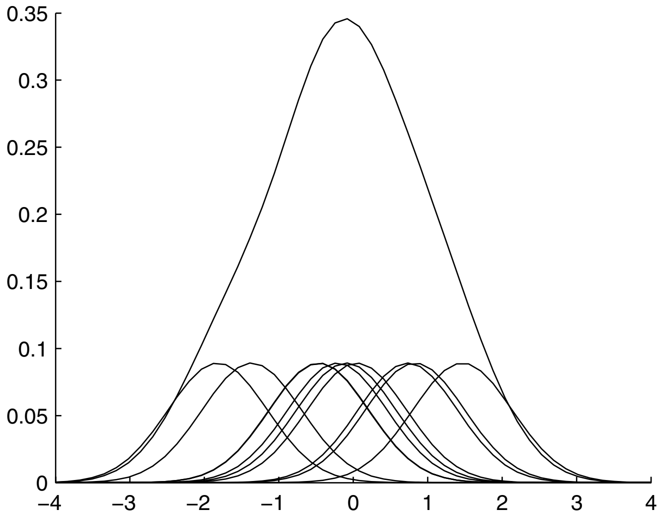

FIGURE 8.8.
We obtain the above kernel density estimate for $n = 10$ random variables. A weighted kernel is centered at each data point, and the curves are averaged together to obtain the estimate. Note that there are two 'bumps' where there is a higher concentration of smaller densities.

Notice that the places where there are more curves or kernels yield *'bumps'* in the final estimate. An alternative implementation is discussed in the exercises.

PROCEDURE - UNIVARIATE KERNEL

1. Choose a kernel, a smoothing parameter h, and the domain (the set of x values) over which to evaluate $\hat{f}(x)$.
2. For each X_i, evaluate the following kernel at all x in the domain:

$$K_i = K\left(\frac{x - X_i}{h}\right); \qquad i = 1, \ldots, n.$$

   The result from this is a set of n curves, one for each data point X_i.
3. Weight each curve by $1/h$.
4. For each x, take the average of the weighted curves.

Example 8.6

In this example, we show how to obtain the kernel density estimate for a data set, using the standard normal density as our kernel. We use the procedure outlined above. The resulting probability density estimate is shown in Figure 8.8.

```
% Generate standard normal random variables.
n = 10;
data = randn(1,n);
% We will get the density estimate at these x values.
x = linspace(-4,4,50);
fhat = zeros(size(x));
h = 1.06*n^(-1/5);
hold on
for i=1:n
   % get each kernel function evaluated at x
   % centered at data
   f = exp(-(1/(2*h^2))*(x-data(i)).^2)/sqrt(2*pi)/h;
   plot(x,f/(n*h));
   fhat = fhat+f/(n);
end
plot(x,fhat);
hold off
```

❑

As in the histogram, the parameter h determines the amount of smoothing we have in the estimate $\hat{f}_{Ker}(x)$. In kernel density estimation, the h is usually called the ***window width***. A small value of h yields a rough curve, while a large value of h yields a smoother curve. This is illustrated in Figure 8.9, where we show kernel density estimates $\hat{f}_{Ker}(x)$ at various window widths. Notice that when the window width is small, we get a lot of noise or spurious structure in the estimate. When the window width is larger we get a smoother estimate, but there is the possibility that we might obscure bumps or other interesting structure in the estimate. In practice, it is recommended that the analyst examine kernel density estimates for different window widths to explore the data and to search for structures such as modes or bumps.

As with the other univariate probability density estimators, we are interested in determining appropriate values for the parameter h. These can be obtained by choosing values for h that minimize the asymptotic MISE. Scott [1992] shows that, under certain conditions, the AMISE for a nonnegative univariate kernel density estimator is

$$\mathrm{AMISE}_{Ker}(h) = \frac{R(K)}{nh} + \frac{1}{4}\sigma_k^4 h^4 R(f''), \tag{8.28}$$

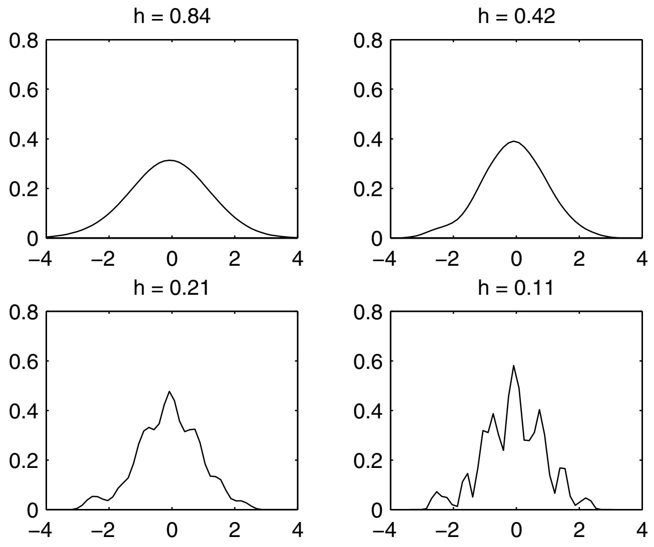

FIGURE 8.9
Four kernel density estimates using $n = 100$ standard normal random variables. Four different window widths are used. Note that as h gets smaller, the estimate gets rougher.

where the kernel K is a continuous probability density function with $\mu_K = 0$ and $0 < \sigma_K^2 < \infty$. The window width that minimizes this is given by

$$h^*_{Ker} = \left(\frac{R(K)}{n\sigma_k^4 R(f'')} \right)^{1/5} . \tag{8.29}$$

Parzen [1962] and Scott [1992] describe the conditions under which this holds. Notice in Equation 8.28 that we have the same bias-variance trade-off with h that we had in previous density estimates.

For a kernel that is equal to the normal density $R(f'') = 3/(8\sqrt{\pi}\sigma^5)$, we have the following Normal Reference Rule for the window width h.

NORMAL REFERENCE RULE - KERNELS

$$h^*_{Ker} = \left(\frac{4}{3}\right)^{1/5} \sigma n^{-1/5} \approx 1.06\sigma n^{-1/5} .$$

We can use some suitable estimate for σ, such as the standard deviation, or $\hat{\sigma} = IQR/1.348$. The latter yields a window width of

$$\hat{h}^*_{Ker} = 0.786 \times IQR \times n^{-1/5}.$$

Silverman [1986] recommends that one use whichever is smaller, the sample standard deviation or $IQR/1.348$ as an estimate for σ.

We now turn our attention to the problem of what kernel to use in our estimate. It is known [Scott, 1992] that the choice of smoothing parameter h is more important than choosing the kernel. This arises from the fact that the effects from the choice of kernel (e.g., kernel tail behavior) are reduced by the averaging process. We discuss the efficiency of the kernels below, but what really drives the choice of a kernel are computational considerations or the amount of differentiability required in the estimate.

In terms of efficiency, the optimal kernel was shown to be [Epanechnikov, 1969]

$$K(t) = \begin{cases} \frac{3}{4}(1-t^2); & -1 \le t \le 1 \\ 0; & \text{otherwise.} \end{cases}$$

It is illustrated in Figure 8.10 along with some other kernels.

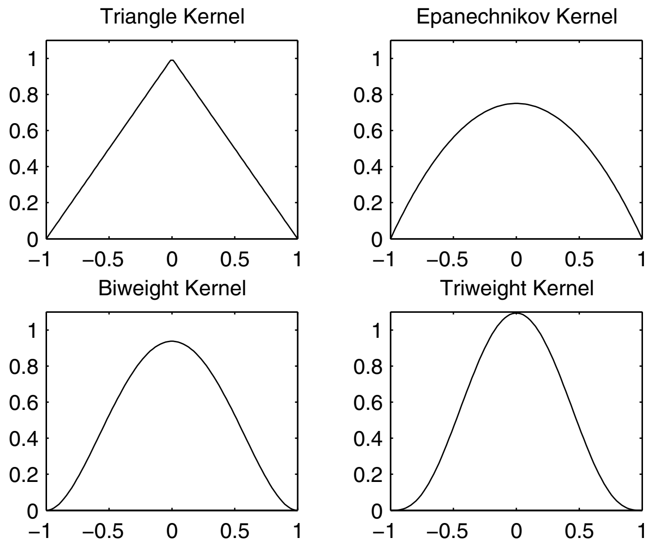

FIGURE 8.10
These illustrate four kernels that can be used in probability density estimation.

Several choices for kernels are given in Table 8.1. Silverman [1986] and Scott [1992] show that these kernels have efficiencies close to that of the Epanechnikov kernel, the least efficient being the normal kernel. Thus, it seems that efficiency should not be the major consideration in deciding what kernel to use. It is recommended that one choose the kernel based on other considerations as stated above.

TABLE 8.1

Examples of Kernels for Density Estimation

Kernel Name	Equation
Triangle	$K(t) = (1 - \lvert t \rvert) \qquad -1 \le t \le 1$
Epanechnikov	$K(t) = \frac{3}{4}(1 - t^2) \qquad -1 \le t \le 1$
Biweight	$K(t) = \frac{15}{16}(1 - t^2)^2 \qquad -1 \le t \le 1$
Triweight	$K(t) = \frac{35}{32}(1 - t^2)^3 \qquad -1 \le t \le 1$
Normal	$K(t) = \frac{1}{\sqrt{2\pi}} \exp\left\{\frac{-t^2}{2}\right\} \qquad -\infty < t < \infty$

Multivariate Kernel Estimators

Here we assume that we have a sample of size n, where each observation is a d-dimensional vector, $\mathbf{X}_i, i = 1, \ldots, n$. The simplest case for the multivariate kernel estimator is the product kernel. Descriptions of the general kernel density estimate can be found in Scott [1992] and in Silverman [1986]. The ***product kernel*** is

$$\hat{f}_{Ker}(\mathbf{x}) = \frac{1}{nh_1 \ldots h_d} \sum_{i=1}^{n} \left\{ \prod_{j=1}^{d} K\left(\frac{x_j - X_{ij}}{h_j}\right) \right\}, \tag{8.30}$$

where X_{ij} is the j-th component of the i-th observation. Note that this is the product of the same univariate kernel, with a (possibly) different window width in each dimension. Since the product kernel estimate is comprised of univariate kernels, we can use any of the kernels that were discussed previously.

Scott [1992] gives expressions for the asymptotic integrated squared bias and asymptotic integrated variance for the multivariate product kernel. If the normal kernel is used, then minimizing these yields a normal reference rule for the multivariate case, which is given below.

NORMAL REFERENCE RULE - KERNEL (MULTIVARIATE)

$$h^*_{j_{Ker}} = \left(\frac{4}{n(d+2)}\right)^{\frac{1}{d+4}} \sigma_j; \qquad j = 1, \ldots, d,$$

where a suitable estimate for σ_j can be used. If there is any skewness or kurtosis evident in the data, then the window widths should be narrower, as discussed previously. The skewness factor for the frequency polygon (Equation 8.20) can be used here.

Example 8.7

In this example, we construct the product kernel estimator for the **iris** data. To make it easier to visualize, we use only the first two variables (sepal length and sepal width) for each species. So, we first create a data matrix comprised of the first two columns for each species.

```
load iris
% Create bivariate data matrix with all three species.
data = [setosa(:,1:2)];
data(51:100,:) = versicolor(:,1:2);
data(101:150,:) = virginica(:,1:2);
```

Next we obtain the smoothing parameter using the Normal Reference Rule.

```
% Get the window width using the Normal Ref Rule.
[n,p] = size(data);
s = sqrt(var(data));
hx = s(1)*n^(-1/6);
hy = s(2)*n^(-1/6);
```

The next step is to create a grid over which we will construct the estimate.

```
% Get the ranges for x and y & construct grid.
num_pts = 30;
minx = min(data(:,1));
maxx = max(data(:,1));
miny = min(data(:,2));
maxy = max(data(:,2));
```

```
gridx = ((maxx+2*hx)-(minx-2*hx))/num_pts
gridy = ((maxy+2*hy)-(miny-2*hy))/num_pts
[X,Y]=meshgrid((minx-2*hx):gridx:(maxx+2*hx),...
     (miny-2*hy):gridy:(maxy+2*hy));
x = X(:);   %put into col vectors
y = Y(:);
```

We are now ready to get the estimates. Note that in this example, we are changing the form of the loop. Instead of evaluating each weighted curve and then averaging, we will be looping over each point in the domain.

```
z = zeros(size(x));
for i=1:length(x)
   xloc = x(i)*ones(n,1);
   yloc = y(i)*ones(n,1);
   argx = ((xloc-data(:,1))/hx).^2;
   argy = ((yloc-data(:,2))/hy).^2;
   z(i) = (sum(exp(-.5*(argx+argy))))/(n*hx*hy*2*pi);
end
[mm,nn] = size(X);
Z = reshape(z,mm,nn);
```

We show the surface plot for this estimate in Figure 8.11. As before, we can verify that our estimate is a *bona fide* by estimating the area under the curve. In this example, we get an area of 0.9994.

```
area = sum(sum(Z))*gridx*gridy;
```

❑

Before leaving this section, we present a summary of univariate probability density estimators and their corresponding Normal Reference Rule for the smoothing parameter h. These are given in Table 8.2.

8.4 Finite Mixtures

So far, we have been discussing nonparametric density estimation methods that require a choice of smoothing parameter h. In the previous section, we showed that we can get different estimates of our probability density depending on our choice for h. It would be helpful if we could avoid choosing a smoothing parameter. In this section, we present a method called finite mixtures that does not require a smoothing parameter. However, as is often the case, when we eliminate one parameter we end up replacing it with another. In finite mixtures, we do not have to worry about the smoothing parameter. Instead, we have to determine the number of terms in the mixture.

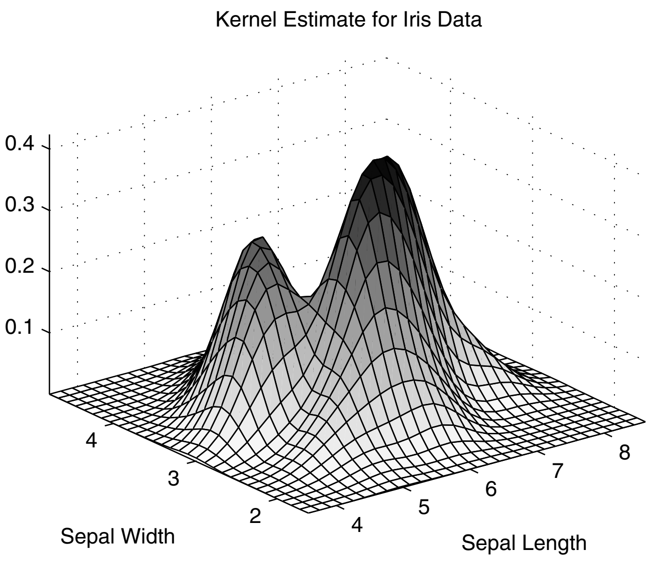

FIGURE 8.11
This is the product kernel density estimate for the sepal length and sepal width of the **iris** data. These data contain all three species. The presence of peaks in the data indicate that two of the species might be distinguishable based on these two variables.

TABLE 8.2

Summary of Univariate Probability Density Estimators and the Normal Reference Rule for the Smoothing Parameter

Method	Estimator	Normal Reference Rule
Histogram	$\hat{f}_{Hist}(x) = \frac{v_k}{nh}$ x in B_k	$h^*_{Hist} = 3.5\sigma n^{-1/3}$
Frequency Polygon	$\hat{f}_{FP}(x) = \left(\frac{1}{2} - \frac{x}{h}\right)\hat{f}_k + \left(\frac{1}{2} + \frac{x}{h}\right)\hat{f}_{k+1}$ $\bar{B}_K \leq x \leq \bar{B}_{k+1}$	$h^*_{FP} = 2.15\sigma n^{-1/5}$
Kernel	$\hat{f}_{Ker}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$	$h^*_{Ker} = 1.06\sigma n^{-1/5}$; K is the normal kernel.

Finite mixtures offer advantages in the area of the computational load put on the system. Two issues to consider with many probability density estimation methods are the computational burden in terms of the amount of information we have to store and the computational effort needed to obtain the probability density estimate at a point. We can illustrate these ideas using the kernel density estimation method. To evaluate the estimate at a point x (in the univariate case) we have to retain all of the data points, because the estimate is a weighted sum of n kernels centered at each sample point. In addition, we must calculate the value of the kernel n times. The situation for histograms and frequency polygons is a little better. The amount of information we must store to provide an estimate of the probability density is essentially driven by the number of bins. Of course, the situation becomes worse when we move to multivariate kernel estimates, histograms, and frequency polygons. With the massive, high-dimensional data sets we often work with, the computational effort and the amount of information that must be stored to use the density estimates is an important consideration. Finite mixtures is a technique for estimating probability density functions that can require relatively little computer storage space or computations to evaluate the density estimates.

Univariate Finite Mixtures

The finite mixture method assumes the density $f(x)$ can be modeled as the sum of c weighted densities, with $c << n$. The most general case for the univariate finite mixture is

$$f(x) = \sum_{i=1}^{c} p_i g(x;\theta_i), \tag{8.31}$$

where p_i represents the ***weight*** or ***mixing coefficient*** for the i-th term, and $g(x;\theta_i)$ denotes a probability density, with parameters represented by the vector θ_i. To make sure that this is a *bona fide* density, we must impose the condition that $p_1 + \ldots + p_c = 1$ and $p_i > 0$. To evaluate $f(x)$, we take our point x, find the value of the component densities $g(x;\theta_i)$ at that point, and take the weighted sum of these values.

Example 8.8

The following example shows how to evaluate a finite mixture model at a given x. We construct the curve for a three term finite mixture model, where the component densities are taken to be normal. The model is given by

$$f(x) = 0.3 \times \phi(x;-3, 1) + 0.3 \times \phi(x;0, 1) + 0.4 \times \phi(x;2, 0.5),$$

where $\phi(x;\mu, \sigma^2)$ represents the normal probability density function at x. We see from the model that we have three terms or component densities, centered at -3, 0, and 2. The mixing coefficient or weight for the first two terms are 0.3 leaving a weight of 0.4 for the last term. The following MATLAB code produces the curve for this model and is shown in Figure 8.12.

```
% Create a domain x for the mixture.
x = linspace(-6,5);
% Create the model - normal components used.
mix = [0.3 0.3 0.4];        % mixing coefficients
mus = [-3 0 2];             % term means
vars = [1 1 0.5];
nterm = 3;
% Use Statistics Toolbox function to evaluate
% normal pdf.
fhat = zeros(size(x));
for i = 1:nterm
   fhat = fhat+mix(i)*normpdf(x,mus(i),vars(i));
end
plot(x,fhat)
title('3 Term Finite Mixture')
```

❑

Hopefully, the reader can see the connection between finite mixtures and kernel density estimation. Recall that in the case of univariate kernel density estimators, we obtain these by evaluating a weighted kernel centered at each sample point, and adding these n terms. So, a kernel estimate can be considered a special case of a finite mixture where $c = n$.

The component densities of the finite mixture can be any probability density function, continuous or discrete. In this book, we confine our attention to the continuous case and use the normal density for the component function. Therefore, the estimate of a finite mixture would be written as

$$\hat{f}_{FM}(x) = \sum_{i=1}^{c} \hat{p}_i \phi(x;\hat{\mu}_i, \hat{\sigma}_i^2), \tag{8.32}$$

where $\phi(x;\hat{\mu}_i, \hat{\sigma}_i^2)$ denotes the normal probability density function with mean $\hat{\mu}_i$ and variance $\hat{\sigma}_i^2$. In this case, we have to estimate c-1 independent mixing coefficients, as well as the c means and c variances using the data. Note that to evaluate the density estimate at a point x, we only need to retain these $3c - 1$ parameters. Since $c << n$, this can be a significant computational savings over evaluating density estimates using the kernel method. With finite mixtures much of the computational burden is shifted to the estimation part of the problem.

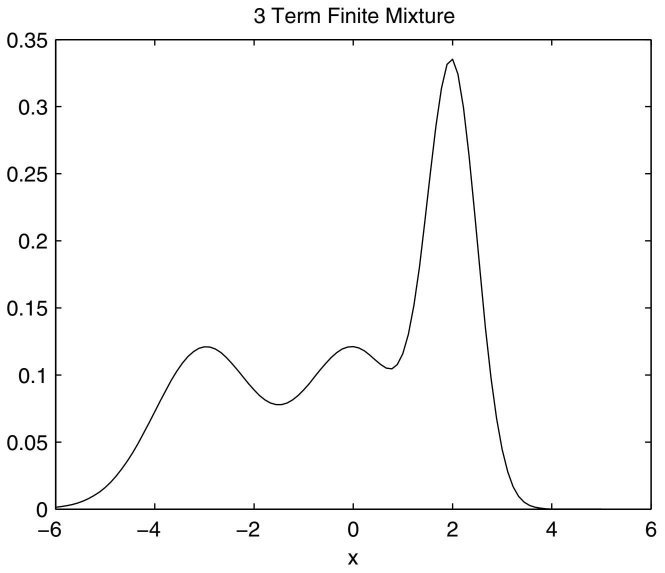

FIGURE 8.12
This shows the probability density function corresponding to the three-term finite mixture model from Example 8.8.

Visualizing Finite Mixtures

The methodology used to estimate the parameters for finite mixture models will be presented later on in this section (page 296). We first show a method for visualizing the underlying structure of finite mixtures with normal component densities [Priebe, et al. 1994], because it is used to help visualize and explain another approach to density estimation (adaptive mixtures). Here, structure refers to the number of terms in the mixture, along with the component means and variances. In essence, we are trying to visualize the high-dimensional parameter space (recall there are $3c$-1 parameters for the univariate mixture of normals) in a 2-D representation. This is called a *dF* plot, where each component is represented by a circle. The circles are centered at the mean μ_i and the mixing coefficient p_i. The size of the radius of the circle indicates the standard deviation. An example of a *dF* plot is given in Figure 8.13 and is discussed in the following example.

Example 8.9

We construct a *dF* plot for the finite mixture model discussed in the previous example. Recall that the model is given by

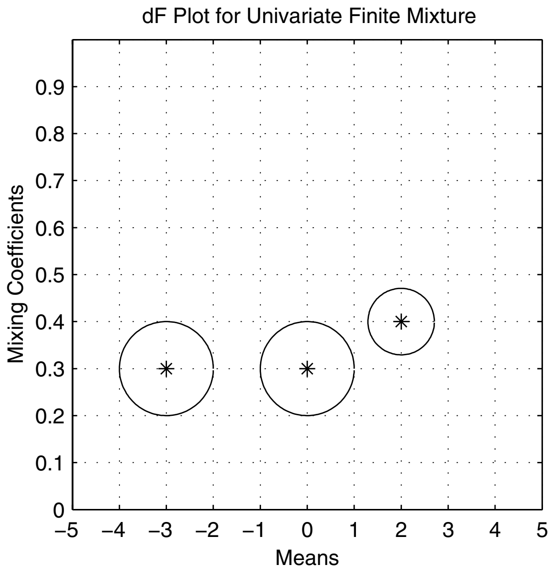

FIGURE 8.13
This shows the *dF* plot for the three term finite mixture model of Figure 8.12.

$$f(x) = 0.3 \times \phi(x;-3, 1) + 0.3 \times \phi(x;0, 1) + 0.4 \times \phi(x;2, 0.5) .$$

Our first step is to set up the model consisting of the number of terms, the component parameters and the mixing coefficients.

```
% Recall the model - normal components used.
mix = [0.3 0.3 0.4];        % mixing coefficients
mus = [-3 0 2];             % term means
vars = [1 1 0.5];
nterm = 3;
```

Next we set up the figure for plotting. Note that we re-scale the mixing coefficients for easier plotting on the vertical axis and then map the labels to the corresponding value.

```
t = 0:.05:2*pi+eps;  % values to create circle
% To get some scales right.
minx = -5;
maxx = 5;
scale = maxx-minx;
lim = [minx maxx minx maxx];
% Set up the axis limits.
```

```
figure
axis equal
axis(lim)
grid on
% Create and plot a circle for each term.
hold on
for i=1:nterm
   % rescale for plotting purposes
   ycord = scale*mix(i)+minx;
   xc = mus(i)+sqrt(vars(i))*cos(t);
   yc = ycord+sqrt(vars(i))*sin(t);
   plot(xc,yc,mus(i),ycord,'*')
end
hold off
% Relabel the axis to show the right coefficient.
tick = (maxx-minx)/10;
set(gca,'Ytick',minx:tick:maxx)
set(gca,'XTick',minx:tick:maxx)
set(gca,'YTickLabel',...
  '0|0.1|0.2|0.3|0.4|0.5|0.6|0.7|0.8|0.9|1')
xlabel('Means'),ylabel('Mixing Coefficients')
title('dF Plot for Univariate Finite Mixture')
```

The first circle on the left corresponds to the component with $p_i = 0.3$ and $\mu_i = -3$. Similarly, the middle circle of Figure 8.13 represents the second term of the model. Note that this representation of the mixture makes it easier to see which terms carry more weight and where they are located in the domain.
❑

Multivariate Finite Mixtures

Finite mixtures is easily extended to the multivariate case. Here we define the multivariate finite mixture model as the weighted sum of multivariate component densities,

$$f(\mathbf{x}) = \sum_{i=1}^{c} p_i g(\mathbf{x}; \theta_i) .$$

As before, the mixing coefficients or weights must be nonnegative and sum to one, and the component density parameters are represented by θ_i. When we are estimating the function, we often use the multivariate normal as the component density. This gives the following equation for an estimate of a multivariate finite mixture

$$\hat{f}_{FM}(\mathbf{x}) = \sum_{i=1}^{c} \hat{p}_i \phi(x; \hat{\mu}_i, \hat{\Sigma}_i), \tag{8.33}$$

where $\mathbf{x}$ is a d-dimensional vector, $\hat{\mu}_i$ is a d-dimensional vector of means, and $\hat{\Sigma}_i$ is a $d \times d$ covariance matrix. There are still c-1 mixing coefficients to estimate. However, there are now $c \times d$ values that have to be estimated for the means and $(cd(c+1))/2$ values for the component covariance matrices.

The *dF* representation has been extended [Solka, Poston, Wegman, 1995] to show the structure of a multivariate finite mixture, when the data are 2-D or 3-D. In the 2-D case, we represent each term by an ellipse centered at the mean of the component density $\hat{\mu}_i$, with the eccentricity of the ellipse showing the covariance structure of the term. For example, a term with a covariance that is close to the identity matrix will be shown as a circle. We label the center of each ellipse with text identifying the mixing coefficient. An example is illustrated in Figure 8.14.

A *dF* plot for a trivariate finite mixture can be fashioned by using color to represent the values of the mixing coefficients. In this case, we use the three dimensions in our plot to represent the means for each term. Instead of ellipses, we move to ellipsoids, with eccentricity determined by the covariance as before. See Figure 8.15 for an example of a trivariate *dF* plot. The *dF* plots are particularly useful when working with the adaptive mixtures density estimation method that will be discussed shortly. We provide a function called **`csdfplot`** that will implement the *dF* plots for univariate, bivariate and trivariate data.

Example 8.10

In this example, we show how to implement the function called **`csdfplot`** and illustrate its use with bivariate and trivariate models. The bivariate case is the following three component model:

$$p_1 = 0.5 \qquad p_2 = 0.3 \qquad p_3 = 0.2,$$

$$\mu_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \qquad \mu_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mu_3 = \begin{bmatrix} 5 \\ 6 \end{bmatrix},$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \Sigma_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \qquad \Sigma_3 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}.$$

```
% First create the model.
% The function expects a vector of weights;
% a matrix of means, where each column of the matrix
% corresponds to a d-D mean; a 3-D array of
% covariances, where each page of the array is a
% covariance matrix.
pies = [0.5 0.3 0.2]; % mixing coefficients
mus = [-1 1 5; -1 1 6];
% Delete any previous variances in the workspace.
clear vars
vars(:,:,1) = eye(2);
vars(:,:,2) = eye(2)*.5
vars(:,:,3) = [1 0.5; 0.5 1];
figure
csdfplot(mus,vars,pies)
```

The resulting plot is shown in Figure 8.14. Note that the covariance of two of the component densities are represented by circles, with one larger than the other. These correspond to the first two terms of the model. The third component density has an elliptical covariance structure indicating non-zero off-diagonal elements in the covariance matrix. We now do the same thing for the trivariate case, where the model is

$$\mu_1 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \qquad \mu_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mu_3 = \begin{bmatrix} 5 \\ 6 \\ 2 \end{bmatrix},$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \Sigma_2 = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \qquad \Sigma_3 = \begin{bmatrix} 1 & 0.7 & 0.2 \\ 0.7 & 1 & 0.5 \\ 0.2 & 0.5 & 1 \end{bmatrix}.$$

The mixing coefficients are the same as before. We need only to adjust the means and the covariance accordingly.

```
mus(3,:) = [-1 1 2];
% Delete previous vars array or you will get an error.
clear vars
vars(:,:,1) = eye(3);
vars(:,:,2) = eye(3)*.5;
vars(:,:,3)=[1 0.7 0.2;
             0.7 1 0.5;
             0.2 0.5 1];
figure
csdfplot(mus,vars,pies)
% get a different viewpoint
view([-34,9])
```

The trivariate *dF* plot for this model is shown in Figure 8.15. Two terms (the first two) are shown as spheres and one as an ellipsoid.
❑

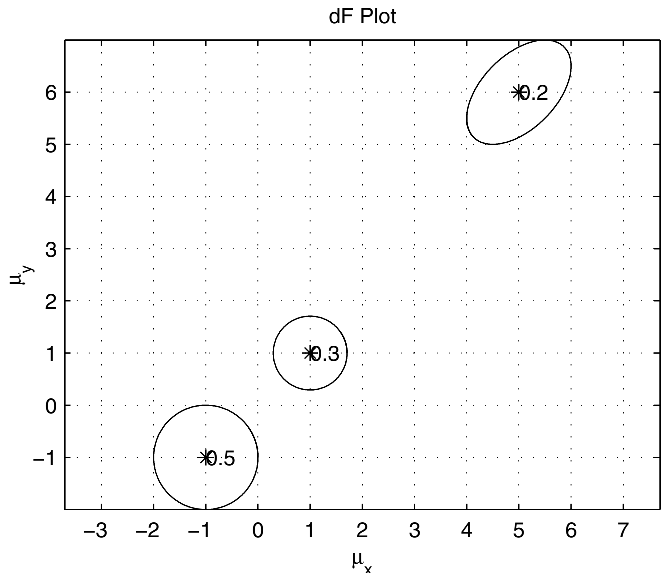

FIGURE 8.14
Bivariate *dF* plot for the three term mixture model of Example 8.10.

EM Algorithm for Estimating the Parameters

The problem of estimating the parameters in a finite mixture has been studied extensively in the literature. The book by Everitt and Hand [1981] provides an excellent overview of this topic and offers several methods for parameter estimation. The technique we present here is called the Expectation-Maximization (EM) method. This is a general method for optimizing likelihood functions and is useful in situations where data might be missing or simpler optimization methods fail. The seminal paper on this topic is by Dempster, Laird and Rubin [1977], where they formalize the EM algorithm and establish its properties. Redner and Walker [1984] apply it to mixture densities. The EM methodology is now a standard tool for statisticians and is used in many applications.

In this section, we discuss the EM algorithm as it can be applied to estimating the parameters of a finite mixture of normal densities. To use the EM algo-

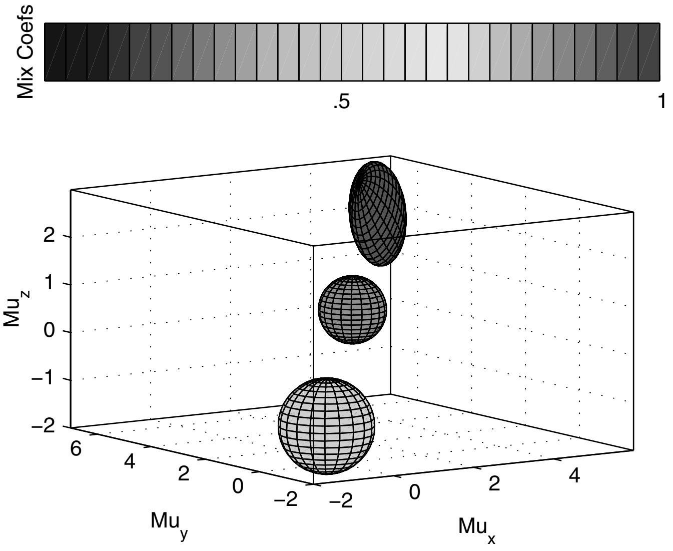

FIGURE 8.15
Trivariate *dF* plot for the three term mixture model of Example 8.10.

rithm, we must have a value for the number of terms c in the mixture. This is usually obtained using prior knowledge of the application (the analyst expects a certain number of groups), using graphical exploratory data analysis (looking for clusters or other group structure) or using some other method of estimating the number of terms. The approach called adaptive mixtures [Priebe, 1994] offers a way to address the problem of determining the number of component densities to use in the finite mixture model. This approach is discussed later.

Besides the number of terms, we must also have an initial guess for the value of the component parameters. Once we have an initial estimate, we update the parameter estimates using the data and the equations given below. These are called the iterative EM update equations, and we provide the multivariate case as the most general one. The univariate case follows easily.

The first step is to determine the posterior probabilities given by

$$\hat{\tau}_{ij} = \frac{\hat{p}_i \phi(\mathbf{x}_j; \hat{\mu}_i, \hat{\Sigma}_i)}{\hat{f}(\mathbf{x}_j)}; \qquad i = 1, \ldots, c\,;\ j = 1, \ldots, n\,. \tag{8.34}$$

where $\hat{\tau}_{ij}$ represents the estimated posterior probability that point $\mathbf{x}_j$ belongs to the i-th term, $\phi(\mathbf{x}_j;\hat{\mu}_i, \hat{\Sigma}_i)$ is the multivariate normal density for the i-th term evaluated at $\mathbf{x}_j$, and

$$\hat{f}(\mathbf{x}_j) = \sum_{k=1}^{c} \hat{p}_k \phi(\mathbf{x}_j;\hat{\mu}_k, \hat{\Sigma}_k) \tag{8.35}$$

is the finite mixture estimate at point $\mathbf{x}_j$.

The posterior probability tells us the likelihood that a point belongs to each of the separate component densities. We can use this estimated posterior probability to obtain a weighted update of the parameters for each component. This yields the iterative EM update equations for the mixing coefficients, the means and the covariance matrices. These are

$$\hat{p}_i = \frac{1}{n}\sum_{j=1}^{n} \hat{\tau}_{ij} \tag{8.36}$$

$$\hat{\mu}_i = \frac{1}{n}\sum_{j=1}^{n} \frac{\hat{\tau}_{ij}\mathbf{x}_j}{\hat{p}_i} \tag{8.37}$$

$$\hat{\Sigma}_i = \frac{1}{n}\sum_{j=1}^{n} \frac{\hat{\tau}_{ij}(\mathbf{x}_j - \hat{\mu}_i)(\mathbf{x}_j - \hat{\mu}_i)^T}{\hat{p}_i}. \tag{8.38}$$

Note that if $d = 1$, then the update equation for the variance is

$$\hat{\sigma}_i^2 = \frac{1}{n}\sum_{j=1}^{n} \frac{\hat{\tau}_{ij}(x_j - \hat{\mu}_i)^2}{\hat{p}_i}. \tag{8.39}$$

The steps for the EM algorithm to estimate the parameters for a finite mixture with multivariate normal components are given here and are illustrated in Example 8.11.

FINITE MIXTURES - EM PROCEDURE

1. Determine the number of terms or component densities c in the mixture.

2. Determine an initial guess at the component parameters. These are the mixing coefficients, means and covariance matrices for each normal density.
3. For each data point $\mathbf{x}_j$, calculate the posterior probability using Equation 8.34.
4. Update the mixing coefficients, the means and the covariance matrices for the individual components using Equations 8.36 through 8.38.
5. Repeat steps 3 through 4 until the estimates converge.

Typically, step 5 is implemented by continuing the iteration until the changes in the estimates at each iteration are less than some pre-set tolerance. Note that with the iterative EM algorithm, we need to use the entire data set to simultaneously update the parameter estimates. This imposes a high computational load when dealing with massive data sets.

Example 8.11

In this example, we provide the MATLAB code that implements the multivariate EM algorithm for estimating the parameters of a finite mixture probability density model. To illustrate this, we will generate a data set that is a mixture of two terms with equal mixing coefficients. One term is centered at the point $(-2, 2)$ and the other is centered at $(2, 0)$. The covariance of each component density is given by the identity matrix. Our first step is to generate 200 data points from this distribution.

```
% Create some artificial two-term mixture data.
n = 200;
data = zeros(n,2);
% Now generate 200 random variables. First find
% the number that come from each component.
r = rand(1,n);
% Find the number generated from component 1.
ind = length(find(r <= 0.5));
% Create some mixture data. Note that the
% component densities are multivariate normals.
% Generate the first term.
data(1:ind,1) = randn(ind,1) - 2;
data(1:ind,2) = randn(ind,1) + 2;
% Generate the second term.
data(ind+1:n,1) = randn(n-ind,1) + 2;
data(ind+1:n,2) = randn(n-ind,1);
```

We must then specify various parameters for the EM algorithm, such as the number of terms.

```
c = 2;   % number of terms
```

```
[n,d] = size(data);  % n=# pts, d=# dims
tol = 0.00001;   % set up criterion for stopping EM
max_it = 100;
totprob = zeros(n,1);
```

We also need an initial guess at the component density parameters.

```
% Get the initial parameters for the model to start EM
mu(:,1) = [-1 -1]';  % each column represents a mean
mu(:,2) = [1 1]';
mix_cof = [0.3 0.7];
var_mat(:,:,1) = eye(d);
var_mat(:,:,2) = eye(d);
varup = zeros(size(var_mat));
muup = zeros(size(mu));
% Just to get started.
num_it = 1;
deltol = tol+1;% to get started
```

The following steps implement the EM update formulas found in Equations 8.34 through 8.38.

```
while num_it <= max_it & deltol > tol
   % get the posterior probabilities
   totprob = zeros(n,1);
   for i=1:c
     posterior(:,i) = mix_cof(i)*...
         csevalnorm(data,mu(:,i)',var_mat(:,:,i));
      totprob = totprob+posterior(:,i);
   end
   den = totprob*ones(1,c);
   posterior = posterior./den;
   % Update the mixing coefficients.
   mix_cofup = sum(posterior)/n;
   % Update the means.
   mut = data'*posterior;
   MIX = ones(d,1)*mix_cof;
   muup = mut./(MIX*n);
   % Update the means and the variances.
   for i=1:c
      cen_data = data-ones(n,1)*mu(:,i)';
      mat = cen_data'*...
         diag(posterior(:,i))*cen_data;
      varup(:,:,i)=mat./(mix_cof(i)*n);
   end
   % Get the tolerances.
   delvar = max(max(max(abs(varup-var_mat))));
   delmu = max(max(abs(muup-mu)));
```

```
      delpi = max(abs(mix_cof-mix_cofup));
      deltol = max([delvar,delmu,delpi]);
      % Reset parameters.
      num_it = num_it+1;
      mix_cof = mix_cofup;
      mu = muup;
      var_mat = varup;
   end  % while loop
```

For our data set, it took 37 iterations to converge to an answer. The convergence of the EM algorithm to a solution and the number of iterations depends on the tolerance, the initial parameters, the data set, etc. The estimated model returned by the EM algorithm is

$$\hat{p}_1 = 0.498 \qquad \hat{p}_2 = 0.502\,,$$

$$\hat{\mu}_1 = \begin{bmatrix} -2.08 \\ 2.03 \end{bmatrix} \qquad \hat{\mu}_2 = \begin{bmatrix} 1.83 \\ -0.03 \end{bmatrix}.$$

For brevity, we omit the estimated covariances, but we can see from these results that the model does match the data that we generated.
❑

Adaptive Mixtures

The adaptive mixtures [Priebe, 1994] method for density estimation uses a data-driven approach for estimating the number of component densities in a mixture model. This technique uses the recursive EM update equations that are provided below. The basic idea behind adaptive mixtures is to take one point at a time and determine the distance from the observation to each component density in the model. If the distance to each component is larger than some threshold, then a new term is created. If the distance is less than the threshold for all terms, then the parameter estimates are updated based on the recursive EM equations.

We start our explanation of the adaptive mixtures approach with a description of the recursive EM algorithm for mixtures of multivariate normal densities. This method recursively updates the parameter estimates based on a new observation. As before, the first step is to determine the posterior probability that the new observation belongs to each term:

$$\hat{\tau}_i^{(n+1)} = \frac{\hat{p}_i^{(n)}\phi(\mathbf{x}^{(n+1)};\hat{\mu}_i^{(n)},\hat{\Sigma}_i^{(n)})}{\hat{f}^{(n)}(\mathbf{x}^{(n+1)})}; \qquad i = 1, \ldots, c\,, \tag{8.40}$$

where $\hat{\tau}_i^{(n+1)}$ represents the estimated posterior probability that the new observation $\mathbf{x}^{(n+1)}$ belongs to the i-th term, and the superscript (n) denotes the estimated parameter values based on the previous n observations. The denominator is the finite mixture density estimate

$$\hat{f}^{(n)}(\mathbf{x}^{(n+1)}) = \sum_{i=1}^{c} \hat{p}_i \phi(\mathbf{x}^{(n+1)}; \hat{\mu}_i^{(n)}, \hat{\Sigma}_i^{(n)})$$

for the new observation using the mixture from the previous n points.

The remainder of the recursive EM update equations are given by Equations 8.41 through 8.43. Note that recursive equations are typically in the form of the old value for an estimate plus an update term using the new observation. The recursive update equations for mixtures of multivariate normals are:

$$\hat{p}_i^{(n+1)} = \hat{p}_i^{(n)} + \frac{1}{n}(\hat{\tau}_i^{(n+1)} - \hat{p}_i^{(n)}) \tag{8.41}$$

$$\hat{\mu}_i^{(n+1)} = \hat{\mu}_i^{(n)} + \frac{\hat{\tau}_i^{(n+1)}}{n\hat{p}_i^{(n)}}(\mathbf{x}^{(n+1)} - \hat{\mu}_i^{(n)}) \tag{8.42}$$

$$\hat{\Sigma}_i^{(n+1)} = \hat{\Sigma}_i^{(n)} + \frac{\hat{\tau}_i^{(n+1)}}{n\hat{p}_i^{(n)}}\left[(\mathbf{x}^{(n+1)} - \hat{\mu}_i^{(n)})(\mathbf{x}^{(n+1)} - \hat{\mu}_i^{(n)})^T - \hat{\Sigma}_i^{(n)}\right] . \tag{8.43}$$

This reduces to the 1-D case in a straightforward manner, as was the case with the iterative EM update equations.

The adaptive mixtures approach updates our probability density estimate $\hat{f}(\mathbf{x})$ and also provides the opportunity to expand the parameter space (i.e., the model) if the data indicate that should be done. To accomplish this, we need a way to determine when a new component density should be added. This could be done in several ways, but the one we present here is based on the Mahalanobis distance. If this distance is too large for all of the terms (or alternatively if the minimum distance is larger than some threshold), then we can consider the new point too far away from the existing terms to update the current model. Therefore, we create a new term.

The squared Mahalanobis distance between the new observation $\mathbf{x}^{(n+1)}$ and the i-th term is given by

$$MD_i^2(\mathbf{x}^{(n+1)}) = (\mathbf{x}^{(n+1)} - \hat{\mu}_i^{(n)})^T\left(\hat{\Sigma}_i^{(n)}\right)^{-1}(\mathbf{x}^{(n+1)} - \hat{\mu}_i^{(n)}) . \tag{8.44}$$

We create a new term if

$$\min_i\{MD_i^2(\mathbf{x}^{(n+1)})\} > t_C, \tag{8.45}$$

where t_C is a threshold to create a new term. The rule in Equation 8.45 states that if the smallest squared Mahalanobis distance is greater than the threshold, then we create a new term. In the univariate case, if $t_C = 1$ is used, then a new term is created if a new observation is more than one standard deviation away from the mean of each term. For $t_C = 4$, a new term would be created for an observation that is at least two standard deviations away from the existing terms. For multivariate data, we would like to keep the same term creation rate as in the 1-D case. Solka [1995] provides thresholds t_C based on the squared Mahalanobis distance for the univariate, bivariate, and trivariate cases. These are shown in Table 8.3.

TABLE 8.3

Recommended Thresholds for Adaptive Mixtures

Dimensionality	Create Threshold
1	1
2	2.34
3	3.54

When we create a new term, we initialize the parameters using Equations 8.46 through 8.48. We denote the current number of terms in the model by N.

$$\hat{\mu}_{N+1}^{(n+1)} = \mathbf{x}^{(n+1)}, \tag{8.46}$$

$$\hat{p}_{N+1}^{(n+1)} = \frac{1}{n+1}, \tag{8.47}$$

$$\hat{\Sigma}_{N+1}^{(n+1)} = \Im(\hat{\Sigma}_i), \tag{8.48}$$

where $\Im(\hat{\Sigma}_i)$ is a weighted average using the posterior probabilities. In practice, some other estimate or initial covariance can be used for the new term. To ensure that the mixing coefficients sum to one when a new term is added, the $\hat{p}_i^{(n+1)}$ must be rescaled using

$$\hat{p}_i^{(n+1)} = \frac{n\hat{p}_i^{(n)}}{n+1}; \qquad i = 1, \ldots, N.$$

We continue through the data set, one point at a time, adding new terms as necessary. Our density estimate is then given by

$$\hat{f}_{AM}(\mathbf{x}) = \sum_{i=1}^{N} \hat{p}_i \phi(\mathbf{x}; \hat{\mu}_i, \hat{\Sigma}_i). \tag{8.49}$$

This allows for a variable number of terms N, where usually $N << n$. The adaptive mixtures technique is captured in the procedure given here, and a function called **`csadpmix`** is provided with the Computational Statistics Toolbox. Its use in the univariate case is illustrated in Example 8.12.

ADAPTIVE MIXTURES PROCEDURE:

1. Initialize the adaptive mixtures procedure using the first data point $\mathbf{x}^{(1)}$:

   $$\hat{\mu}_1^{(1)} = \mathbf{x}^{(1)}, \hat{p}_1^{(1)} = 1, \text{ and } \hat{\Sigma}_1^{(1)} = \mathbf{I},$$

   where **I** denotes the identity matrix. In the univariate case, the variance of the initial term is one.
2. For a new data point $\mathbf{x}^{(n+1)}$, calculate the squared Mahalanobis distance as in Equation 8.44.
3. If the minimum squared distance is greater than t_C, then create a new term using Equations 8.46 through 8.48. Increase the number of terms N by one.
4. If the minimum squared distance is less than the create threshold t_C, then update the existing terms using Equations 8.41 through 8.43.
5. Continue steps 2 through 4 using all data points.

In practice, the adaptive mixtures method is used to get initial values for the parameters, as well as an estimate of the number of terms needed to model the density. One would then use these as a starting point and apply the iterative EM algorithm to refine the estimates.

Example 8.12

In this example, we illustrate the MATLAB code that implements the univariate adaptive mixtures density estimation procedure. The source code for these functions are given in Appendix D. We generate random variables using the same three term mixture model that was discussed in Example 8.9.Recall that the model is given by

$$f(x) = 0.3 \times \phi(x;-3, 1) + 0.3 \times \phi(x;0, 1) + 0.4 \times \phi(x;2, 0.5) .$$

```
% Get the true model to generate data.
pi_tru = [0.3 0.3 0.4];
n = 100;
x = zeros(n,1);
% Now generate 100 random variables. First find
% the number that fall in each one.
r = rand(1,100);
% Find the number generated from each component.
ind1 = length(find(r <= 0.3));
ind2 = length(find(r > 0.3 & r <= 0.6));
ind3 = length(find(r > 0.6));
% create some artificial  3 term mixture data
x(1:ind1) = randn(ind1,1) - 3;
x(ind1+1:ind2+ind1)=randn(ind2,1);
x(ind1+ind2+1:n) = randn(ind3,1)*sqrt(0.5)+2;
```

We now call the adaptive mixtures function **csadpmix** to estimate the model.

```
% Now call the adaptive mixtures function.
maxterms = 25;
[pihat,muhat,varhat] = csadpmix(x,maxterms);
```

The following MATLAB commands provide the plots shown in Figure 8.16.

```
% Get the plots.
csdfplot(muhat,varhat,pihat,min(x),max(x));
axis equal
nterms = length(pihat);
figure
csplotuni(pihat,muhat,varhat,...
    nterms,min(x)-5,max(x)+5,100)
```

We reorder the observations and repeat the process to get the plots in Figure 8.17.

```
% Now re-order the points and repeat
% the adaptive mixtures process.
ind = randperm(n);
x = x(ind);
[pihat,muhat,varhat] = csadpmix(x,maxterms);
```

❑

Our example above demonstrates some interesting things to consider with adaptive mixtures. First, the model complexity or the number of terms is sometimes greater than is needed. For example, in Figure 8.16, we show a *dF* plot for the three term mixture model in Example 8.12. Note that the adaptive mixture approach yields more than three terms. This is a problem with mixture models in general. Different models (i.e., number of terms and estimated component parameters) can produce essentially the same function estimate or curve for $\hat{f}(\mathbf{x})$. This is illustrated in Figures 8.16 and 8.17, where we see that similar curves are obtained from two different models for the same data set. These results are straight from the adaptive mixtures density estimation approach. In other words, we did not use this estimate as an initial starting point for the EM approach. If we had applied the iterative EM to these estimated models, then the curves should be the same.

The other issue that must be considered when using the adaptive mixtures approach is that the resulting model or estimated probability density function depends on the order in which the data are presented to the algorithm. This is also illustrated in Figures 8.16 and 8.17, where the second estimated model is obtained after re-ordering the data. These issues were addressed by Solka [1995].

8.5 Generating Random Variables

In the introduction, we discussed several uses of probability density estimates, and it is our hope that the reader will discover many more. One of the applications of density estimation is in the area of modeling and simulation. Recall that a key aspect of modeling and simulation is the collection of data generated according to some underlying random process and the desire to generate more random variables from the same process for simulation purposes. One option is to use one of the density estimation techniques discussed in this chapter and randomly sample from that distribution. In this section, we provide the methodology for generating random variables from finite or adaptive mixtures density estimates.

We have already seen an example of this procedure in Example 8.11 and Example 8.12. The procedure is to first choose the class membership of generated observations based on uniform (0,1) random variables. The number of random variables generated from each component density is given by the corresponding proportion of these uniform variables that are in the required range. The steps are outlined here.

PROCEDURE - GENERATING RANDOM VARIABLES (FINITE MIXTURE)

1. We are given a finite mixture model $(p_i, g_i(\mathbf{x};\theta_i))$ with c components, and we want to generate n random variables from that distribution.

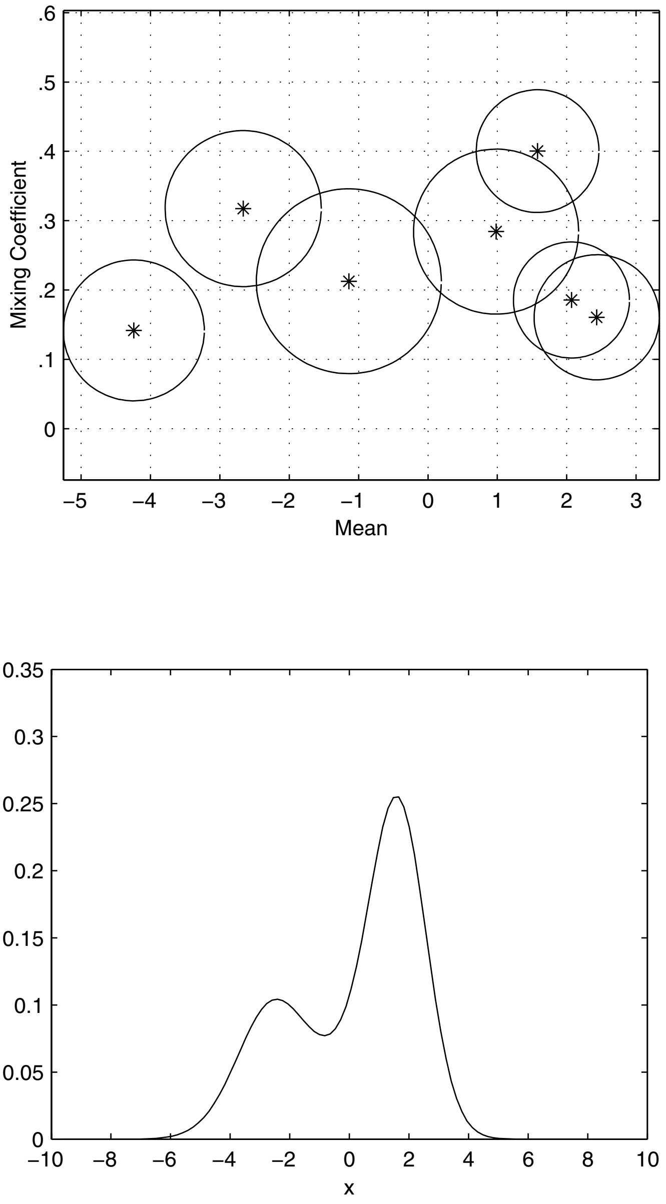

FIGURE 8.16
The upper plot shows the *dF* representation for Example 8.12. Compare this with Figure 8.17 for the same data. Note that the curves are essentially the same, but the number of terms and associated parameters are different. Thus, we can get different models for the same data.

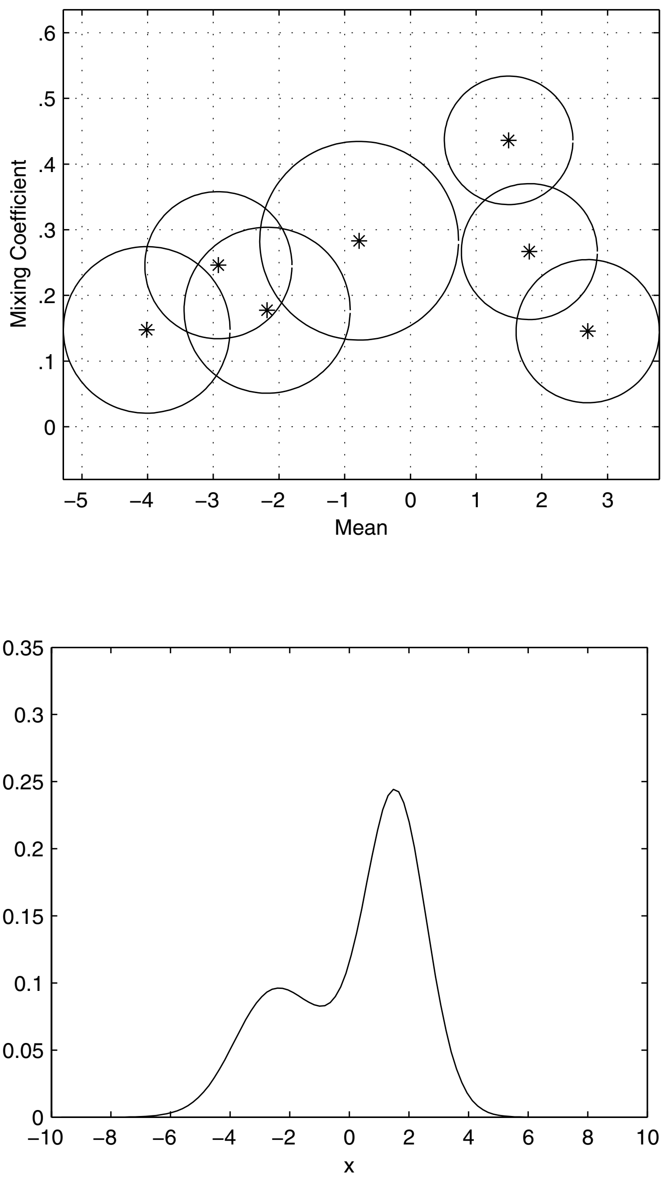

FIGURE 8.17
This is the second estimated model using adaptive mixtures for the data generated in Example 8.12. This second model was obtained by re-ordering the data set and then implementing the adaptive mixtures technique. This shows the dependence of the technique on the order in which the data are presented to the method.

2. First determine the component membership of each of the n random variables. We do this by generating n uniform (0,1) random variables (U_i). Component membership is determined as follows

   If $0 \le U_i < p_1$, then X_i is from component density 1.

   If $p_1 \le U_i < p_1 + p_2$, then X_i is from component density 2.

   . . .

   If $\sum_{j=1}^{c-1} p_j \le U_i \le 1$, then X_i is from component density c.
3. Generate the X_i from the corresponding $g_i(\mathbf{x};\theta_i)$ using the component membership found in step 2.

Note that with this procedure, one could generate random variables from a mixture of any component densities. For instance, the model could be a mixture of exponentials, betas, etc.

Example 8.13

Generate a random sample of size n from a finite mixture estimate of the Old Faithful Geyser data (**`geyser`**). First we have to load up the data and build a finite mixture model.

```
load geyser
% Expects rows to be observations.
data = geyser';
% Get the finite mixture.
% Use a two term model.
% Set initial model to means at 50 and 80.
muin = [50, 80];
% Set mixing coefficients equal.
piesin = [0.5, 0.5];
% Set initial variances to 1.
varin = [1, 1];
max_it = 100;
tol = 0.001;
% Call the finite mixtures.
[pies,mus,vars]=...
    csfinmix(data,muin,varin,piesin,max_it,tol);
```

Now generate some random variables according to this estimated model.

```
% Now generate some random variables from this model.
% Get the true model to generate data from this.
n = 300;
x = zeros(n,1);
```

```
% Now generate 300 random variables. First find
% the number that fall in each one.
r = rand(1,n);
% Find the number generated from component 1.
ind = length(find(r <= pies(1)));
% Create some mixture data. Note that the
% component densities are normals.
x(1:ind) = randn(ind,1)*sqrt(vars(1)) + mus(1);
x(ind+1:n) = randn(n-ind,1)*sqrt(vars(2)) + mus(2);
```

We can plot density histograms to compare the two data sets. These are shown in Figure 8.18. Not surprisingly, they look similar, but different. The user is asked to explore this further in the exercises.
❑

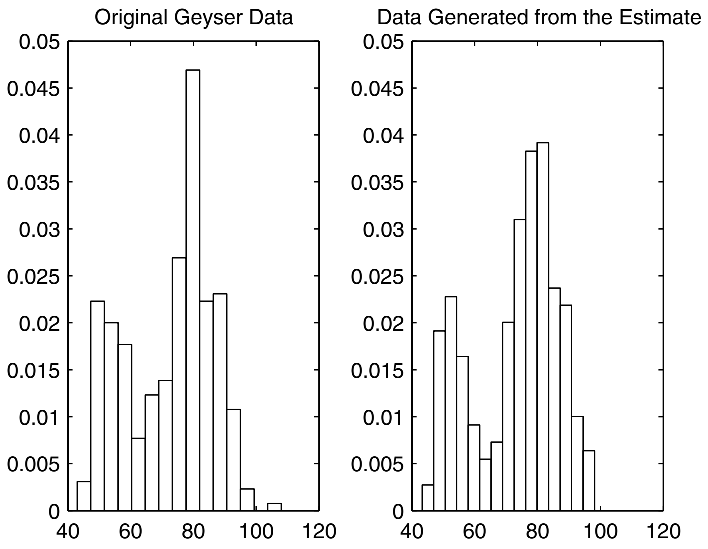

FIGURE 8.18
Histogram density estimates of the Old Faithful geyser data. The one on the right shows the estimate from the data that was sampled from the finite mixture density estimate of the original data.

8.6 MATLAB Code

The MATLAB Statistics Toolbox does not have any functions for nonparametric density estimation. The functions it has for estimating distribution parameters (e.g., **`mle`**, **`normfit`**, **`expfit`**, **`betafit`**, etc.) can be used for parametric density estimation. The standard MATLAB package has functions for frequency histograms, as explained in Chapter 5.

We provide several functions for nonparametric density estimation with the Computational Statistics Toolbox. These are listed in Table 8.4.

TABLE 8.4

List of Functions from Chapter 8 Included in the Computational Statistics Toolbox

Purpose	MATLAB Function
These provide a bivariate histogram.	**`cshist2d`** **`cshistden`**
This returns a frequency polygon density estimate.	**`csfreqpoly`**
This function returns the Averaged Shifted Histogram.	**`csash`**
These functions perform kernel density estimation.	**`cskernnd`** **`cskern2d`**
Create plots	**`csdfplot`** **`csplotuni`**
Functions for finite and adaptive mixtures	**`csfinmix`** **`csadpmix`**

8.7 Further Reading

The discussion of histograms, frequency polygons and averaged shifted histograms presented in this book follows that of Scott [1992]. Scott's book is an excellent resource for univariate and multivariate density estimation, and it describes many applications of the techniques. It includes a comprehensive treatment of the underlying theory on selecting smoothing parameters, analyzing the performance of density estimates in terms of the asymptotic mean integrated squared error, and also addresses high dimensional data.

The summary book by Silverman [1986] provides a relatively non-theoretical treatment of density estimation. He includes a discussion of histograms, kernel methods and others. This book is readily accessible to most statisticians, data analysts or engineers. It contains applications and computational details, making the subject easier to understand.

Other books on density estimation include Tapia and Thompson [1978], Devroye and Gyorfi [1985], Wand and Jones [1995], and Simonoff [1996]. The Tapia and Thompson book offers a theoretical foundation for density estimation and includes a discussion of Monte Carlo simulations. The Devroye and Gyorfi text describes the underlying theory of density estimation using the L_1 (absolute error) viewpoint instead of L_2 (squared error). The books by Wand and Jones and Simonoff look at using kernel methods for smoothing and exploratory data analysis.

A paper by Izenman [1991] provides a comprehensive review of many methods in univariate and multivariate density estimation and includes an extensive bibliography. Besides histograms and kernel methods, he discusses projection pursuit density estimation [Friedman, Stuetzle, and Schroeder, 1984], maximum penalized likelihood estimators, sieve estimators, and orthogonal estimators.

For the reader who would like more information on finite mixtures, we recommend Everitt and Hand [1981] for a general discussion of this topic. The book provides a summary of the techniques for obtaining mixture models (estimating the parameters) and illustrates them using applications. That text also discusses ways to handle the problem of determining the number of terms in the mixture and other methods for estimating the parameters. It is appropriate for someone with a general statistics or engineering background. For readers who would like more information on the theoretical details of finite mixtures, we refer them to McLachlan and Basford [1988] or Titterington, Smith and Makov [1985]. A recent book by McLachlan and Peel [2000] provides many examples of finite mixtures, linking them to machine learning, data mining, and pattern recognition.

The EM algorithm is described in the text by McLachlan and Krishnan [1997]. This offers a unified treatment of the subject, and provides numerous applications of the EM algorithm to regression, factor analysis, medical imaging, experimental design, finite mixtures, and others.

For a theoretical discussion of the adaptive mixtures approach, the reader is referred to Priebe [1993, 1994]. These examine the error in the adaptive mixtures density estimates and its convergence properties. A recent paper by Priebe and Marchette [2000] describes a data-driven method for obtaining parsimonious mixture model estimates. This methodology addresses some of the problems with the adaptive/finite mixtures approach: 1) that adaptive mixtures is not designed to yield a parsimonious model and 2) how many terms or component densities should be used in a finite mixture model.

Solka, Poston, and Wegman [1995] extend the static *dF* plot to a dynamic one. References to MATLAB code are provided in this paper describing a dynamic view of the adaptive mixtures and finite mixtures estimation process in time (i.e., iterations of the EM algorithm).

Exercises

8.1. Create a MATLAB function that will return the value of the histogram estimate for the probability density function. Do this for the 1-D case.

8.2. Generate a random sample of data from a standard normal. Construct a kernel density estimate of the probability density function and verify that the area under the curve is approximately 1 using **trapz**.

8.3. Generate 100 univariate normals and construct a histogram. Calculate the MSE at a point x_0 using Monte Carlo simulation. Do this for varying bin widths. What is the better bin width? Does the sample size make a difference? Does it matter whether x_0 is in the tails or closer to the mean? Repeat this experiment using the absolute error. Are your conclusions similar?

8.4. Generate univariate normal random variables. Using the Normal Reference Rules for h, construct a histogram, a frequency polygon and a kernel estimate of the data. Estimate the MSE at a point x_0 using Monte Carlo simulation.

8.5. Generate a random sample from the exponential distribution. Construct a histogram using the Normal Reference Rule. Using Monte Carlo simulation, estimate the MISE. Use the skewness factor to adjust h and re-estimate the MISE. Which window width is better?

8.6. Use the **snowfall** data and create a MATLAB **movie** that shows how 1-D histograms change with bin width. See **help** on **movie** for information on how to do this. Also make a **movie** showing how changing the bin origin affects the histogram.

8.7. Repeat Example 8.2 for bin widths given by the Freedman-Diaconis Rule. Is there a difference in the results? What does the histogram look like if you use Sturge's Rule?

8.8. Write a MATLAB function that will return the value of a bivariate histogram at a point, given the bin counts, the sample size, and the window widths.

8.9. Write a MATLAB function that will evaluate the cumulative distribution function for a univariate frequency polygon. You can use the **trapz**, **quad**, or **quadl** functions.

8.10. Load the **iris** data. Create a 150×2 matrix by concatenating the first two columns of each species. Construct and plot a frequency polygon of these data. Do the same thing for all possible pairs of columns. You might also look at a **contour** plot of the frequency polygons. Is there evidence of groups in the plots?

8.11. In this chapter, we showed how you could construct a kernel density estimate by placing a weighted kernel at each data point, evaluating the kernels over the domain, and then averaging the *n* curves. In that implementation, we are looping over all of the data points. An alternative implementation is to loop over all points in the domain where you want to get the value of the estimate, evaluate a weighted kernel at each point, and take the average. The following code shows you how to do this. Implement this using the Buffalo **snowfall** data. Verify that this is a valid density by estimating the area under the curve.

```
load snowfall
x = 0:140;
n = length(snowfall);
h = 1.06*sqrt(var(snowfall))*n^(-1/5);
fhat = zeros(size(x));
% Loop over all values of x in the domain
% to get the kernel evaluated at that point.
for i = 1:length(x)
 xloc = x(i)*ones(1,n);
 % Take each value of x and evaluate it at
 % n weighted kernels -
 % each one centered at a data point, then add them up.
 arg = ((xloc-snowfall)/h).^2;
 fhat(i) = (sum(exp(-.5*(arg)))/(n*h*sqrt(2*pi)));
end
```

8.12. Write a MATLAB function that will construct a kernel density estimate for the multivariate case.

8.13. Write a MATLAB function that will provide the finite mixture density estimate at a point in *d* dimensions.

8.14. Implement the univariate adaptive mixtures density estimation procedure on the Buffalo **snowfall** data. Once you have your initial model, use the EM algorithm to refine the estimate.

8.15. In Example 8.13, we generate a random sample from the kernel estimate of the Old Faithful **geyser** data. Repeat this example to obtain a new random sample of **geyser** data from the estimated model and construct a new density estimate from the second sample. Find the integrated squared error between the two density estimates. Does the error between the curves indicate that the second random sample generates a similar density curve?

8.16. Say we have a kernel density estimate where the kernel used is a normal density. If we put this in the context of finite mixtures, then what are the values for the component parameters (p_i, μ_i, σ_i^2) in the corresponding finite mixture?

8.17. Repeat Example 8.12. Plot the curves from the estimated models. What is the ISE between the two estimates? Use the iterative EM algorithm on both models to refine the estimates. What is the ISE after you do this? What can you say about the two different models? Are your conclusions different if you use the IAE?

8.18. Write a MATLAB function that will generate random variables (univariate or multivariate) from a finite mixture of normals.

8.19. Using the method for generating random variables from a finite mixture that was discussed in this chapter, develop and implement an algorithm for generating random variables based on a kernel density estimate.

8.20. Write a function that will estimate the MISE between two functions. Convert it to also estimate the MIAE between two functions.

8.21. Apply some of the univariate density estimation techniques from this chapter to the **`forearm`** data.

8.22. The **`elderly`** data set contains the height measurements (in centimeters) of 351 elderly females [Hand, et al., 1994]. Use some of the univariate density estimation techniques from this chapter to explore the data. Is there evidence of bumps and modes?

8.23. Apply the multivariate techniques of this chapter to the **`nfl`** data [Csorgo and Welsh, 1989; Hand, et al., 1994]. These data contain bivariate measurements of the game time to the first points scored by kicking the ball between the end posts (X_1), and the game time to the first points scored by moving the ball into the end zone (X_2). The times are in minutes and seconds. Plot your results.