Chapter 4
Generating Random Variables

4.1 Introduction

Many of the methods in computational statistics require the ability to generate random variables from known probability distributions. This is at the heart of Monte Carlo simulation for statistical inference (Chapter 6), bootstrap and resampling methods (Chapters 6 and 7), Markov chain Monte Carlo techniques (Chapter 11), and the analysis of spatial point processes (Chapter 12). In addition, we use simulated random variables to explain many other topics in this book, such as exploratory data analysis (Chapter 5), density estimation (Chapter 8), and statistical pattern recognition (Chapter 9).

There are many excellent books available that discuss techniques for generating random variables and the underlying theory; references will be provided in the last section. Our purpose in covering this topic is to give the reader the tools they need to generate the types of random variables that often arise in practice and to provide examples illustrating the methods. We first discuss general techniques for generating random variables, such as the inverse transformation and acceptance-rejection methods. We then provide algorithms and MATLAB code for generating random variables for some useful distributions.

4.2 General Techniques for Generating Random Variables

Uniform Random Numbers

Most methods for generating random variables start with random numbers that are uniformly distributed on the interval \((0, 1)\). We will denote these random variables by the letter \(U\). With the advent of computers, we now have
the ability to generate uniform random variables very easily. However, we have to caution the reader that the numbers generated by computers are really pseudorandom because they are generated using a deterministic algorithm. The techniques used to generate uniform random variables have been widely studied in the literature, and it has been shown that some generators have serious flaws [Gentle, 1998].

The basic MATLAB program has a function \texttt{rand} for generating uniform random variables. There are several optional arguments, and we take a moment to discuss them because they will be useful in simulation. The function \texttt{rand} with no arguments returns a single instance of the random variable \( U \). To get an \( m \times n \) array of uniform variates, you can use the syntax \texttt{rand}(m,n). A note of caution: if you use \texttt{rand}(n), then you get an \( n \times n \) matrix.

The sequence of random numbers that is generated in MATLAB depends on the seed or the state of the generator. The state is reset to the default when it starts up, so the same sequences of random variables are generated whenever you start MATLAB. This can sometimes be an advantage in situations where we would like to obtain a specific random sample, as we illustrate in the next example. If you call the function using \texttt{rand('state',0)}, then MATLAB resets the generator to the initial state. If you want to specify another state, then use the syntax \texttt{rand('state',j)} to set the generator to the \( j \)-th state. You can obtain the current state using \texttt{S = rand('state')}, where \( S \) is a 35 element vector. To reset the state to this one, use \texttt{rand('state',S)}.

It should be noted that random numbers that are uniformly distributed over an interval \( a \) to \( b \) may be generated by a simple transformation, as follows

\[
X = (b - a) \cdot U + a. \tag{4.1}
\]

\textbf{Example 4.1}

In this example, we illustrate the use of MATLAB’s function \texttt{rand}.

\begin{verbatim}
% Obtain a vector of uniform random variables in (0,1).
x = rand(1,1000);
% Do a histogram to plot.
% First get the height of the bars.
[N,X] = hist(x,15);
% Use the bar function to plot.
bar(X,N,1,'w')
title('Histogram of Uniform Random Variables')
xlabel('X')
ylabel('Frequency')
\end{verbatim}

The resulting histogram is shown in Figure 4.1. In some situations, the analyst might need to reproduce results from a simulation, say to verify a con-
% Generate 3 random samples of size 5.
% Allocate the memory.
for i = 1:3
    rand('state',i) % set the state
    x(i,:) = rand(1,5);
end

The three sets of random variables are

0.9528   0.7041   0.9539   0.5982   0.8407
0.8752   0.3179   0.2732   0.6765   0.0712
0.5162   0.2252   0.1837   0.2163   0.4272

We can easily recover the five random variables generated in the second sample by setting the state of the random number generator, as follows

rand('state',2)
x = rand(1,5);

FIGURE 4.1
This figure shows a histogram of a random sample from the uniform distribution on the interval (0, 1).
From this, we get
\[ xt = 0.8752 \quad 0.3179 \quad 0.2732 \quad 0.6765 \quad 0.0712 \]
which is the same as before.

**Inverse Transform Method**

The inverse transform method can be used to generate random variables from a continuous distribution. It uses the fact that the cumulative distribution function \( F \) is uniform (0, 1) [Ross, 1997]:

\[ U = F(X). \tag{4.2} \]

If \( U \) is a uniform \((0, 1)\) random variable, then we can obtain the desired random variable \( X \) from the following relationship

\[ X = F^{-1}(U). \tag{4.3} \]

We see an example of how to use the inverse transform method when we discuss generating random variables from the exponential distribution (see Example 4.6). The general procedure for the inverse transformation method is outlined here.

**PROCEDURE - INVERSE TRANSFORM METHOD (CONTINUOUS)**

1. Derive the expression for the inverse distribution function \( F^{-1}(U) \).
2. Generate a uniform random number \( U \).
3. Obtain the desired \( X \) from \( X = F^{-1}(U) \).

This same technique can be adapted to the discrete case [Banks, 2001]. Say we would like to generate a discrete random variable \( X \) that has a probability mass function given by

\[ P(X = x_i) = p_i; \quad x_0 < x_1 < x_2 < \ldots; \quad \sum_{i} p_i = 1. \tag{4.4} \]

We get the random variables by generating a random number \( U \) and then deliver the random number \( X \) according to the following

\[ X = x_i, \quad \text{if} \quad F(x_{i-1}) < U \leq F(x_i). \tag{4.5} \]
We illustrate this procedure using a simple example.

**Example 4.2**

We would like to simulate a discrete random variable $X$ that has probability mass function given by

\[
P(X = 0) = 0.3, \quad P(X = 1) = 0.2, \quad P(X = 2) = 0.5.
\]

The cumulative distribution function is

\[
F(x) = \begin{cases} 
  0; & x < 0 \\
  0.3; & 0 \leq x < 1 \\
  0.5; & 1 \leq x < 2 \\
  1.0; & 2 \leq x.
\end{cases}
\]

We generate random variables for $X$ according to the following scheme

\[
X = \begin{cases} 
  0; & U \leq 0.3 \\
  1; & 0.3 < U \leq 0.5 \\
  2; & 0.5 < U \leq 1.
\end{cases}
\]

This is easily implemented in MATLAB and is left as an exercise. The procedure is illustrated in Figure 4.2, for the situation where a uniform random variable 0.73 was generated. Note that this would return the variate $x = 2$.

We now outline the algorithmic technique for this procedure. This will be useful when we describe a method for generating Poisson random variables.

**PROCEDURE - INVERSE TRANSFORM (DISCRETE)**

1. Define a probability mass function for $x_i$, $i = 1, \ldots, k$. Note that $k$ could grow infinitely.
2. Generate a uniform random number $U$.
3. If $U \leq p_0$ deliver $X = x_0$.
4. else if $U \leq p_0 + p_1$ deliver $X = x_1$.
5. else if $U \leq p_0 + p_1 + p_2$ deliver $X = x_2$.

© 2002 by Chapman & Hall/CRC
Example 4.3
We repeat the previous example using this new procedure and implement it in MATLAB. We first generate 100 variates from the desired probability mass function.

```matlab
% Set up storage space for the variables.
X = zeros(1,100);
% These are the x's in the domain.
x = 0:2;
% These are the probability masses.
pr = [0.3 0.2 0.5];
% Generate 100 rv's from the desired distribution.
for i = 1:100
    u = rand; % Generate the U.
    if u <= pr(1)
        X(i) = x(1);
    elseif u <= sum(pr(1:2)) % It has to be between 0.3 and 0.5.
        X(i) = x(2);
```
else
    \( X(i) = x(3); \) % It has to be between 0.5 and 1.
end
end

One way to verify that our random variables are from the desired distribution is to look at the relative frequency of each \( x \).

% Find the proportion of each number.
\[
x_0 = \frac{\text{length} \left( \text{find} \left( X == 0 \right) \right)}{100};
\]
\[
x_1 = \frac{\text{length} \left( \text{find} \left( X == 1 \right) \right)}{100};
\]
\[
x_2 = \frac{\text{length} \left( \text{find} \left( X == 2 \right) \right)}{100};
\]

The resulting estimated probabilities are
\[
\hat{P}(x = x_0) = 0.26
\]
\[
\hat{P}(x = x_1) = 0.21
\]
\[
\hat{P}(x = x_2) = 0.53.
\]

These values are reasonable when compared with the desired probability mass values.

**Acceptance-Rejection Method**

In some cases, we might have a simple method for generating a random variable from one density, say \( g(y) \), instead of the density we are seeking. We can use this density to generate from the desired continuous density \( f(x) \). We first generate a random number \( Y \) from \( g(y) \) and accept the value with a probability proportional to the ratio \( f(Y)/g(Y) \).

If we define \( c \) as a constant that satisfies
\[
\frac{f(y)}{g(y)} \leq c; \quad \text{for all } y,
\]  
(4.6)
then we can generate the desired variates using the procedure outlined below. The constant \( c \) is needed because we might have to adjust the height of \( g(y) \) to ensure that it is above \( f(y) \). We generate points from \( cg(y) \), and those points that are inside the curve \( f(y) \) are accepted as belonging to the desired density. Those that are outside are rejected. It is best to keep the number of rejected variates small for maximum efficiency.
PROCEDURE - ACCEPTANCE-REJECTION METHOD (CONTINUOUS)

1. Choose a density \( g(y) \) that is easy to sample from.
2. Find a constant \( c \) such that Equation 4.6 is satisfied.
3. Generate a random number \( Y \) from the density \( g(y) \).
4. Generate a uniform random number \( U \).
5. If

\[
U \leq \frac{f(Y)}{cg(Y)},
\]

then accept \( X = Y \), else go to step 3.

Example 4.4
We shall illustrate the acceptance-rejection method by generating random variables from the beta distribution with parameters \( \alpha = 2 \) and \( \beta = 1 \) [Ross, 1997]. This yields the following probability density function

\[
f(x) = 2x; \quad 0 < x < 1. \quad (4.7)
\]

Since the domain of this density is 0 to 1, we use the uniform distribution for our \( g(y) \). We must find a constant that we can use to inflate the uniform so it is above the desired beta density. This constant is given by the maximum value of the density function, and from Equation 4.7, we see that \( c = 2 \). For more complicated functions, techniques from calculus or the MATLAB function \texttt{fminsearch} may be used. The following MATLAB code generates 100 random variates from the desired distribution. We save both the accepted and the rejected variates for display purposes only.

```matlab
c = 2;   % constant
n = 100;  % Generate 100 random variables.
x = zeros(1,n);  % random variates
xy = zeros(1,n);% corresponding y values
rej = zeros(1,n);% rejected variates
rejy = zeros(1,n); % corresponding y values
irv = 1;  
irej = 1;
while irv <= n
    y = rand(1);  % random number from g(y)
    u = rand(1);  % random number for comparison
    if u <= 2*y/c;
        x(irv) = y;
        xy(irv) = u*c;
    end
end
```

© 2002 by Chapman & Hall/CRC
```
irv = irv + 1
else
    rej(irej) = y;
    rejy(irej) = u * c;  % really comparing u * c <= 2 * y
    irej = irej + 1
end
end
```

In Figure 4.3, we show the accepted and rejected random variates that were generated in this process. Note that the accepted variates are those that are less than \( f(x) \).

We can easily adapt this method to generate random variables from a discrete distribution. Here we have a method for simulating a random variable with a probability mass function \( q_i = P(Y = i) \), and we would like to obtain a random variable \( X \) having a probability mass function \( p_i = P(X = i) \). As in the continuous case, we generate a random variable \( Y \) from \( q_i \) and accept this value with probability \( p_i / (cq_i) \).

FIGURE 4.3
This shows the points that were accepted ('o') as being generated by \( f(x) = 2x \) and those points that were rejected ('*'). The curve represents \( f(x) \), so we see that the accepted variates are the ones below the curve.
PROCEDURE - REJECTION METHOD (DISCRETE)

1. Choose a probability mass function \( q_i \) that is easy to sample from.
2. Find a constant \( c \) such that \( p_Y < cq_Y \).
3. Generate a random number \( Y \) from the density \( q_i \).
4. Generate a uniform random number \( U \).
5. If

\[
U \leq \frac{p_Y}{cq_Y},
\]

then deliver \( X = Y \), else go to step 3.

Example 4.5
In this example, we use the discrete form of the acceptance-rejection method to generate random variables according to the probability mass function defined as follows

\[
P(X = 1) = 0.15, \\
P(X = 2) = 0.22, \\
P(X = 3) = 0.33, \\
P(X = 4) = 0.10, \\
P(X = 5) = 0.20.
\]

We let \( q_Y \) be the discrete uniform distribution on \( 1, \ldots, 5 \), where the probability mass function is given by

\[
q_y = \frac{1}{5}; \quad y = 1, \ldots, 5.
\]

We describe a method for generating random variables from the discrete uniform distribution in a later section. The value for \( c \) is obtained as the maximum value of \( p_y/q_y \), which is 1.65. This quantity is obtained by taking the maximum \( p_y \), which is \( P(X = 3) = 0.33 \), and dividing by 1/5:

\[
\frac{\max(p_y)}{1/5} = 0.33 \times 5 = 1.65.
\]

The steps for generating the variates are:
1. Generate a variate $Y$ from the discrete uniform density on 1, ..., 5.
   (One could use the MATLAB Statistics Toolbox function `unidrnd` or `csdunrnd`.)
2. Generate a uniform random number $U$.
3. If
   \[
   U \leq \frac{P_Y}{cq_Y} = \frac{p_Y}{1.65 \cdot 1/5} = \frac{p_Y}{0.33},
   \]
   then deliver $X = Y$, else return to step 1.

The implementation of this example in MATLAB is left as an exercise.

4.3 Generating Continuous Random Variables

Normal Distribution

The main MATLAB program has a function that will generate numbers from
the standard normal distribution, so we do not discuss any techniques for
generating random variables from the normal distribution. For the reader
who is interested in how normal random variates can be generated, most of
the references provided in Section 4.6 contain this information.

The MATLAB function for generating standard normal random variables
is called `randn`, and its functionality is similar to the function `rand` that was
discussed in the previous section. As with the uniform random variable $U$,
we can obtain a normal random variable $X$ with mean $\mu$ and variance $\sigma^2$ by
means of a transformation. Letting $Z$ represent a standard normal random
variable (possibly generated from `randn`), we get the desired $X$ from the rela-
tionship

\[
X = Z \cdot \sigma + \mu. \tag{4.8}
\]

Exponential Distribution

The inverse transform method can be used to generate random variables
from the exponential distribution and serves as an example of this procedure.
The distribution function for an exponential random variable with parameter
$\lambda$ is given by
Letting

\[ u = F(x) = 1 - e^{-\lambda x}, \]  

we can solve for \( x \), as follows

\[
\begin{align*}
  u &= 1 - e^{-\lambda x} \\
  e^{-\lambda x} &= 1 - u \\
  -\lambda x &= \log(1 - u) \\
  x &= -\frac{1}{\lambda} \log(1 - u). 
\end{align*}
\]

By making note of the fact that \( 1 - u \) is also uniformly distributed over the interval \((0,1)\), we can generate exponential random variables with parameter \( \lambda \) using the transformation

\[ X = -\frac{1}{\lambda} \log(U). \]  

**Example 4.6**
The following MATLAB code will generate exponential random variables for a given \( \lambda \).

```matlab
% Set up the parameters.
lam = 2;
n = 1000;
% Generate the random variables.
uni = rand(1,n);
X = -log(uni)/lam;
```

We can generate a set of random variables and plot them to verify that the function does yield exponentially distributed random variables. We plot a histogram of the results along with the theoretical probability density function in Figure 4.4. The MATLAB code given below shows how we did this.

```matlab
% Get the values to draw the theoretical curve.
x = 0:.1:5;
% This is a function in the Statistics Toolbox.
y = exppdf(x,1/2);
% Get the information for the histogram.
[N,h] = hist(X,10);
% Change bar heights to make it correspond to
```

\[ F(x) = 1 - e^{-\lambda x}; \quad 0 < x < \infty. \]  

(4.9)
% the theoretical density - see Chapter 5.
N = N/(h(2)-h(1))/n;
% Do the plots.
bar(h,N,1,'w')
hold on
plot(x,y)
hold off
xlabel('X')
ylabel('f(x) - Exponential')

\begin{figure}
\centering
\includegraphics[width=\textwidth]{image.png}
\caption{This shows a probability density histogram of the random variables generated in Example 4.6. We also superimpose the curve corresponding to the theoretical probability density function with \( \lambda = 2 \). The histogram and the curve match quite well.}
\end{figure}

\textbf{Gamma}

In this section, we present an algorithm for generating a gamma random variable with parameters \((t, \lambda)\), where \(t\) is an integer. Recall that it has the following distribution function
The inverse transform method cannot be used in this case, because a simple closed form solution for its inverse is not possible. It can be shown [Ross, 1997] that the sum of \( t \) independent exponentials with the same parameter \( \lambda \) is a gamma random variable with parameters \( t \) and \( \lambda \). This leads to the following transformation based on \( t \) uniform random numbers,

\[
X = -\frac{1}{\lambda} \log U_1 - \cdots - \frac{1}{\lambda} \log U_t.
\]  

(4.13)

We can simplify this and compute only one logarithm by using a familiar relationship of logarithms. This yields the following

\[
X = -\frac{1}{\lambda} \log (U_1 \times \cdots \times U_t) = -\frac{1}{\lambda} \log \left( \prod_{i=1}^{t} U_i \right).
\]  

(4.14)

**Example 4.7**

The MATLAB code given below implements the algorithm described above for generating gamma random variables, when the parameter \( t \) is an integer.

```matlab
n = 1000;
t = 3;
lam = 2;
% Generate the uniforms needed. Each column
% contains the t uniforms for a realization of a
% gamma random variable.
U = rand(t,n);
% Transform according to Equation 4.13.
% See Example 4.8 for an illustration of Equation 4.14.
logU = -log(U)/lam;
X = sum(logU);

To see whether the implementation of the algorithm is correct, we plot them in a probability density histogram.

% Now do the histogram.
[N,h] = hist(X,10);
% Change bar heights.
N = N/(h(2)-h(1))/n;
% Now get the theoretical probability density.
% This is a function in the Statistics Toolbox.
x = 0:.1:6;
```
\[ y = \text{gampdf}(x, t, 1/lam); \]
\[ \text{bar}(h, N, 1, 'w') \]
\[ \text{hold on} \]
\[ \text{plot}(x, y, 'k') \]
\[ \text{hold off} \]

The histogram and the corresponding theoretical probability density function are shown in Figure 4.5.

![Histogram and theoretical PDF](image)

**FIGURE 4.5**
This shows the probability density histogram for a set of gamma random variables with \( t = 3 \) and \( \lambda = 2 \).

**Chi-Square**

A chi-square random variable with \( \nu \) degrees of freedom is a special case of the gamma distribution, where \( \lambda = 1/2 \), \( t = \nu/2 \) and \( \nu \) is a positive integer. This can be generated using the gamma distribution method described above with one change. We have to make this change, because the method we presented for generating gamma random variables is for integer \( t \), which works for even values of \( \nu \).

When \( \nu \) is even, say \( 2k \), we can obtain a chi-square random variable from
When \( v \) is odd, say \( 2k + 1 \), we can use the fact that the chi-square distribution with \( v \) degrees of freedom is the sum of \( v \) squared independent standard normals [Ross, 1997]. We obtain the required random variable by first simulating a chi-square with \( 2k \) degrees of freedom and adding a squared standard normal variate \( Z \), as follows

\[
X = Z^2 - 2\log \left( \prod_{i=1}^{k} U_i \right). \tag{4.16}
\]

**Example 4.8**

In this example, we provide a function that will generate chi-square random variables.

```matlab
% function X = cschirnd(nu)
% This function will return n chi-square random variables with degrees of freedom nu.
function X = cschirnd(nu)
    % Generate the uniforms needed.
    rm = rem(nu,2);
    k = floor(nu/2);
    if rm == 0 % then even degrees of freedom
        U = rand(k,n);
        if k ~= 1
            X = -2*log(prod(U));
        else
            X = -2*log(U);
        end
    else % odd degrees of freedom
        U = rand(k,n);
        Z = randn(1,n);
        if k ~= 1
            X = Z.^2 - 2*2*log(prod(U));
        else
            X = Z.^2 - 2*2*log(U);
        end
    end

The use of this function to generate random variables is left as an exercise.
```

© 2002 by Chapman & Hall/CRC
The chi-square distribution is useful in situations where we need to systematically investigate the behavior of a statistic by changing the skewness of the distribution. As the degrees of freedom for a chi-square increases, the distribution changes from being right skewed to one approaching normality and symmetry.

**Beta**

The beta distribution is useful in simulations because it covers a wide range of distribution shapes, depending on the values of the parameters $\alpha$ and $\beta$. These shapes include skewed, uniform, approximately normal, and a bimodal distribution with an interior dip.

First, we describe a simple approach for generating beta random variables with parameters $\alpha$ and $\beta$, when both are integers [Rubinstein, 1981; Gentle, 1998]. It is known [David, 1981] that the $k$-th order statistic of $n$ uniform (0,1) variates is distributed according to a beta distribution with parameters $k$ and $n - k + 1$. This means that we can generate random variables from the beta distribution using the following procedure.

**PROCEDURE - BETA RANDOM VARIABLES (INTEGER PARAMETERS)**

1. Generate $\alpha + \beta - 1$ uniform random numbers: $U_1, \ldots, U_{\alpha + \beta - 1}$
2. Deliver $X = U_{\alpha}$ which is the $\alpha$-th order statistic.

One simple way to generate random variates from the beta distribution is to use the following result from Rubinstein [1981]. If $Y_1$ and $Y_2$ are independent random variables, where $Y_1$ has a gamma distribution with parameters $\alpha$ and 1, and $Y_2$ follows a gamma distribution with parameters $\beta$ and 1, then

$$X = \frac{Y_1}{Y_1 + Y_2}$$  \hspace{1cm} (4.17)

is from a beta distribution with parameters $\alpha$ and $\beta$. This is the method that is used in the MATLAB Statistics Toolbox function `betarnd` that generates random variates from the beta distribution. We illustrate the use of `betarnd` in the following example.

**Example 4.9**

We use this example to illustrate the use of the MATLAB Statistics Toolbox function that generates beta random variables. In general, most of these toolbox functions for generating random variables use the following general syntax:

```matlab
rvs = pdfrnd(par1,par2,nrow,ncol);
```
Here, \texttt{pdf} refers to the type of distribution (see Table 4.1, on page 106). The first several arguments represent the appropriate parameters of the distribution, so the number of them might change. The last two arguments denote the number of rows and the number of columns in the array of random variables that are returned by the function. We use the function \texttt{betarnd} to generate random variables from two beta distributions with different parameters $\alpha$ and $\beta$. First we look at the case where $\alpha = 3$ and $\beta = 3$. So, to generate $n = 500$ beta random variables (that are returned in a row vector), we use the following commands:

\begin{verbatim}
% Let $a = 3$, $b = 3$
 n = 500;
 a = 3;
 b = 3;
 rvs = betarnd(a,b,1,n);
\end{verbatim}

We can construct a histogram of the random variables and compare it to the corresponding beta probability density function. This is easily accomplished in MATLAB as shown below.

\begin{verbatim}
% Now do the histogram.
 [N,h] = hist(rvs,10);
% Change bar heights.
 N = N/(h(2)-h(1))/n;
% Now get the theoretical probability density.
 x = 0:.05:1;
 y = betapdf(x,a,b);
 plot(x,y)
 axis equal
 bar(h,N,1,'w')
 hold on
 plot(x,y,'k')
 hold off
\end{verbatim}

The result is shown in the left plot of Figure 4.6. Notice that this density looks approximately bell-shaped. The beta density on the right has parameters $\alpha = 0.5$ and $\beta = 0.5$. We see that this curve has a dip in the middle with modes on either end. The reader is asked to construct this plot in the exercises.

\section*{Multivariate Normal}

In the following chapters, we will have many applications where we need to generate multivariate random variables in order to study the algorithms of computational statistics as they apply to multivariate distributions. Thus, we need some methods for generating multivariate random variables. The easi-
Chapter 4: Generating Random Variables

The method is similar to the one used to generate random variables from a univariate normal distribution. One starts with a $d$-dimensional vector of standard normal random numbers. These can be transformed to the desired distribution using

$$\mathbf{x} = \mathbf{R}^T \mathbf{z} + \mathbf{\mu}.$$  \hspace{1cm} (4.18)

Here $\mathbf{z}$ is a $d \times 1$ vector of standard normal random numbers, $\mathbf{\mu}$ is a $d \times 1$ vector representing the mean, and $\mathbf{R}$ is a $d \times d$ matrix such that $\mathbf{R}^T \mathbf{R} = \Sigma$. The matrix $\mathbf{R}$ can be obtained in several ways, one of which is the Cholesky factorization of the covariance matrix $\Sigma$. This is the method we illustrate below. Another possibility is to factor the matrix using singular value decomposition, which will be shown in the examples provided in Chapter 5.

**Figure 4.6**  
This figure shows two histograms created from random variables generated from the beta distribution. The beta distribution on the left has parameters $\alpha = 3$ and $\beta = 3$, while the one on the right has parameters $\alpha = 0.5$ and $\beta = 0.5$. 

*est distribution of this type to generate is the multivariate normal. We cover other methods for generating random variables from more general multivariate distributions in Chapter 11.*
Example 4.10
The function `csmvrnd` generates multivariate normal random variables using the Cholesky factorization. Note that we are transposing the transformation given in Equation 4.18, yielding the following

\[ X = ZR + \mu^T, \]

where \( X \) is an \( n \times d \) matrix of \( d \)-dimensional random variables and \( Z \) is an \( n \times d \) matrix of standard normal random variables.

```matlab
function X = csmvrnd(mu,covm,n)

% This function will return n multivariate random
% normal variables with d-dimensional mean mu and
% covariance matrix covm. Note that the covariance
% matrix must be positive definite (all eigenvalues
% are greater than zero), and the mean
% vector is a column

d = length(mu);
% Get Cholesky factorization of covariance.
R = chol(covm);
% Generate the standard normal random variables.
Z = randn(n,d);
X = Z*R + ones(n,1)*mu';
```

We illustrate its use by generating some multivariate normal random variables with \( \mu^T = (-2, 3) \) and covariance

\[ \Sigma = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}, \]

```matlab
% Generate the multivariate random normal variables.
mu = [-2;3];
covm = [1 0.7 ; 0.7 1];
X = csmvrnd(mu,covm,500);
```

To check the results, we plot the random variables in a scatterplot in Figure 4.7. We can also calculate the sample mean and sample covariance matrix to compare with what we used as input arguments to `csmvrnd`. By typing `mean(X)` at the command line, we get

\[-2.0629 \quad 2.9394\]

Similarly, entering `corrcoef(X)` at the command line yields
We see that these values for the sample statistics correspond to the desired mean and covariance. We note that you could also use the \texttt{cov} function to compare the variances.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure4.7}
\caption{This shows the scatter plot of the random variables generated using the function \texttt{csmvrnd}.}
\end{figure}

\textbf{Generating Variates on a Sphere}

In some applications, we would like to generate $d$-dimensional random variables that are distributed on the surface of the unit hypersphere $S^d$, $d = 2, \ldots$. Note that when $d = 2$ the surface is a circle, and for $d = 3$ the surface is a sphere. We will be using this technique in Chapter 5, where we present an algorithm for exploratory data analysis using projection pursuit. The easiest method is to generate $d$ standard normal random variables and then to scale them such that the magnitude of the vector is one. This is illustrated in the following example.
Example 4.11
The following function cssphrnd generates random variables on a $d$-dimensional unit sphere. We illustrate its use by generating random variables that are on the unit circle $S^2$.

```matlab
% function X = cssphrnd(n,d);
% This function will generate n d-dimensional random variates that are distributed on the unit d-dimensional sphere. d >= 2

function X = cssphrnd(n,d)
if d < 2
    error('ERROR - d must be greater than 1.')
    break
end
% Generate standard normal random variables.
tmp = randn(d,n);
% Find the magnitude of each column.
% Square each element, add and take the square root.
mag = sqrt(sum(tmp.^2));
% Make a diagonal matrix of them - inverses.
dm = diag(1./mag);
% Multiply to scale properly.
% Transpose so X contains the observations.
X = (tmp*dm)';
```

We can use this function to generate a set of random variables for $d = 2$ and plot the result in Figure 4.8.

```matlab
X = cssphrnd(500,2);
plot(X(:,1),X(:,2),'x')
axis equal
xlabel('X_1'),ylabel('X_2')
```

4.4 Generating Discrete Random Variables

**Binomial**
A binomial random variable with parameters $n$ and $p$ represents the number of successes in $n$ independent trials. We can obtain a binomial random vari-
able by generating \( n \) uniform random numbers \( U_1, U_2, \ldots, U_n \) and letting \( X \) be the number of \( U_i \) that are less than or equal to \( p \). This is easily implemented in MATLAB as illustrated in the following example.

**Example 4.12**
We implement this algorithm for generating binomial random variables in the function `csbinrnd`.

```matlab
% function X = csbinrnd(n,p,N)
% This function will generate N binomial random variables with parameters n and p.

function X = csbinrnd(n,p,N)
X = zeros(1,N);
% Generate the uniform random numbers:
% N variates of n trials.
U = rand(N,n);
% Loop over the rows, finding the number % less than p
for i = 1:N
    ind = find(U(i,:) <= p);
    X(i) = length(ind);
end
```

![Figure 4.8](image.png)

*FIGURE 4.8*  
This is the scatter plot of the random variables generated in Example 4.11. These random variables are distributed on the surface of a 2-D unit sphere (i.e., a unit circle).
We use this function to generate a set of random variables that are distributed according to the binomial distribution with parameters $n = 6$ and $p = 0.5$. The histogram of the random variables is shown in Figure 4.9. Before moving on, we offer the following more efficient way to generate binomial random variables in MATLAB:

\[
X = \text{sum}(\text{rand}(n,N) <= p);
\]

This is the histogram for the binomial random variables generated in Example 4.12. The parameters for the binomial are $n = 6$ and $p = 0.5$.

**Poisson**

We use the inverse transform method for discrete random variables as described in Ross [1997] to generate variates from the Poisson distribution. We need the following recursive relationship between successive Poisson probabilities

\[
p_{i+1} = P(X = i) = \frac{\lambda}{i + 1} p_i; \quad i \geq 0.
\]
This leads to the following algorithm.

**PROCEDURE - GENERATING POISSON RANDOM VARIABLES**

1. Generate a uniform random number \( U \).
2. Initialize the quantities: \( i = 0, p_0 = e^{-\lambda}, \) and \( F_0 = p_0 \).
3. If \( U \leq F_i \), then deliver \( X = i \). Return to step 1.
4. Else increment the values: \( p_{i+1} = \lambda p_i / (i + 1), i = i + 1 \), and \( F_{i+1} = F_i + p_{i+1} \).
5. Return to step 3.

This algorithm could be made more efficient when \( \lambda \) is large. The interested reader is referred to Ross [1997] for more details.

**Example 4.13**

The following shows how to implement the procedure for generating Poisson random variables in MATLAB.

```matlab
% function X = cspoirnd(lam, n)
% This function will generate Poisson
% random variables with parameter lambda.
% The reference for this is Ross, 1997, page 50.

function x = cspoirnd(lam,n)
    x = zeros(1,n);
    j = 1;
    while j <= n
        flag = 1;
        % initialize quantities
        u = rand(1);
        i = 0;
        p = exp(-lam);
        F = p;
        while flag % generate the variate needed
            if u <= F % then accept
                x(j) = i;
                flag = 0;
                j = j+1;
            else % move to next probability
                p = lam*p/(i+1);
                i = i+1;
                F = F + p;
            end
        end
    end
```

© 2002 by Chapman & Hall/CRC
end

We can use this to generate a set of Poisson random variables with \( \lambda = 0.5 \), and show a histogram of the data in Figure 4.10.

```matlab
% Set the parameter for the Poisson.
lam = .5;
N = 500; % Sample size
x = cspoirnd(lam,N);
edges = 0:max(x);
f = histc(x,edges);
bar(edges,f/N,1,'w')
```

As an additional check to ensure that our algorithm is working correctly, we can determine the observed relative frequency of each value of the random variable \( X \) and compare that to the corresponding theoretical values.

```matlab
% Determine the observed relative frequencies.
% These are the estimated values.
relf = zeros(1,max(x)+1);
for i = 0:max(x)
    relf(i+1) = length(find(x==i))/N;
end
% Use the Statistics Toolbox function to get the % theoretical values.
y = poisspdf(0:4,.5);
```

When we print these to the MATLAB command window, we have the following

```matlab
% These are the estimated values.
relf = 0.5860  0.3080  0.0840  0.0200  0.0020
% These are the theoretical values.
y = 0.6065  0.3033  0.0758  0.0126  0.0016
```

---

**Discrete Uniform**

When we implement some of the Monte Carlo methods in Chapter 6 (such as the bootstrap), we will need the ability to generate numbers that follow the discrete uniform distribution. This is a distribution where \( X \) takes on values in the set \( \{1, 2, \ldots, N\} \), and the probability that \( X \) equals any of the numbers is \( 1/N \). This distribution can be used to randomly sample without replacement from a group of \( N \) objects.

We can generate from the discrete uniform distribution using the following transform
Chapter 4: Generating Random Variables

105

where the function \( \lceil y \rceil \), \( y \geq 0 \) means to round up the argument \( y \). The next example shows how to implement this in MATLAB.

Example 4.14

The method for generating discrete uniform is implemented in the function \texttt{csdunrnd}, given below.

```matlab
function X = csdunrnd(N,n)
% This function will generate random variables
% from the discrete uniform distribution. It picks
% numbers uniformly between 1 and N.

X = ceil(N*rand(1,n));
```

To verify that we are generating the right random variables, we can look at the observed relative frequencies. Each should have relative frequency of \( 1/N \). This is shown below where \( N = 5 \) and the sample size is 500.

```matlab
N = 5;
n = 500;
x = csdunrnd(N,n);
```

\( X = \lceil NU \rceil \),

where the function \( \lceil y \rceil \), \( y \geq 0 \) means to round up the argument \( y \).

\( \lambda = 0.5 \).

\[ \begin{array}{c}
\text{FIGURE 4.10} \\
\text{This is the histogram for random variables generated from the Poisson with } \lambda = 0.5. \\
\end{array} \]
% Determine the estimated relative frequencies.
relf = zeros(1,N);
for i = 1:N
    relf(i) = length(find(x==i))/n;
end

Printing out the observed relative frequencies, we have
relf = 0.1820 0.2080 0.2040 0.1900 0.2160

which is close to the theoretical value of \(1/N = 1/5 = 0.2\).

### 4.5 MATLAB Code

The MATLAB Statistics Toolbox has functions that will generate random variables from all of the distributions discussed in Section 2.6. As we explained in that section, the analyst must keep in mind that probability distributions are often defined differently, so caution should be exercised when using any software package. Table 4.1 provides a partial list of the MATLAB functions that are available for random number generation. A complete list can be found in Appendix E. As before, the reader should note that the \texttt{gamrnd}, \texttt{weibrnd}, and \texttt{exprnd} functions use the alternative definition for the given distribution (see 24).

<table>
<thead>
<tr>
<th>Distribution</th>
<th>MATLAB Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta</td>
<td>\texttt{betarnd}</td>
</tr>
<tr>
<td>Binomial</td>
<td>\texttt{binornd}</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>\texttt{chi2rnd}</td>
</tr>
<tr>
<td>Discrete Uniform</td>
<td>\texttt{unidrnd}</td>
</tr>
<tr>
<td>Exponential</td>
<td>\texttt{exprnd}</td>
</tr>
<tr>
<td>Gamma</td>
<td>\texttt{gamrnd}</td>
</tr>
<tr>
<td>Normal</td>
<td>\texttt{normrnd}</td>
</tr>
<tr>
<td>Poisson</td>
<td>\texttt{poissrnd}</td>
</tr>
<tr>
<td>Continuous Uniform</td>
<td>\texttt{unifrnd}</td>
</tr>
<tr>
<td>Weibull</td>
<td>\texttt{weibrnd}</td>
</tr>
</tbody>
</table>

© 2002 by Chapman & Hall/CRC
Another function that might prove useful in implementing computational statistics methods is called \texttt{randperm}. This is provided with the standard MATLAB software package, and it generates random permutations of the integers 1 to \(n\). The result can be used to permute the elements of a vector. For example, to permute the elements of a vector \(x\) of size \(n\), use the following MATLAB statements:

\begin{verbatim}
% Get the permuted indices.
ind = randperm(n);
% Now re-order based on the permuted indices.
xperm = x(ind);
\end{verbatim}

We also provide some functions in the Computational Statistics Toolbox for generating random variables. These are outlined in Table 4.2. Note that these generate random variables using the distributions as defined in Chapter 2.

\begin{table}
\centering
\begin{tabular}{|l|l|}
\hline
Distribution & MATLAB Function \\
\hline
Beta & \texttt{csbetarnd} \\
Binomial & \texttt{csbinrnd} \\
Chi-Square & \texttt{cschirnd} \\
Discrete Uniform & \texttt{csdunrnd} \\
Exponential & \texttt{csexprnd} \\
Gamma & \texttt{csgamrnd} \\
Multivariate Normal & \texttt{csmvnrnd} \\
Poisson & \texttt{cspoirnd} \\
Points on a sphere & \texttt{cssphrnd} \\
\hline
\end{tabular}
\caption{List of Functions from Chapter 4 Included in the Computational Statistics Toolbox}
\end{table}

4.6 Further Reading

In this text we do not attempt to assess the computational efficiency of the methods for generating random variables. If the statistician or engineer is performing extensive Monte Carlo simulations, then the time it takes to generate random samples becomes important. In these situations, the reader is encouraged to consult Gentle [1998] or Rubinstein [1981] for efficient algorithms. Our goal is to provide methods that are easily implemented using MATLAB or other software, in case the data analyst must write his own functions for generating random variables from non-standard distributions.
There has been considerable research into methods for random number generation, and we refer the reader to the sources mentioned below for more information on the theoretical foundations. The book by Ross [1997] is an excellent resource and is suitable for advanced undergraduate students. He addresses simulation in general and includes a discussion of discrete event simulation and Markov chain Monte Carlo methods. Another text that covers the topic of random number generation and Monte Carlo simulation is Gentle [1998]. This book includes an extensive discussion of uniform random number generation and covers more advanced topics such as Gibbs sampling. Two other resources on random number generation are Rubinstein [1981] and Kalos and Whitlock [1986]. For a description of methods for generating random variables from more general multivariate distributions, see Johnson [1987]. The article by Deng and Lin [2000] offers improvements on some of the standard uniform random number generators.

A recent article in the MATLAB News & Notes [Spring, 2001] describes the method employed in MATLAB for obtaining normally distributed random variables. The algorithm that MATLAB uses for generating uniform random numbers is described in a similar newsletter article and is available for download at:

Exercises

4.1. Repeat Example 4.3 using larger sample sizes. What happens to the estimated probability mass function (i.e., the relative frequencies from the random samples) as the sample size gets bigger?

4.2. Write the MATLAB code to implement Example 4.5. Generate 500 random variables from this distribution and construct a histogram (hist function) to verify your code.

4.3. Using the algorithm implemented in Example 4.3, write a MATLAB function that will take any probability mass function (i.e., a vector of probabilities) and return the desired number of random variables generated according to that probability function.

4.4. Write a MATLAB function that will return random numbers that are uniformly distributed over the interval (a, b).

4.5. Write a MATLAB function that will return random numbers from the normal distribution with mean $\mu$ and variance $\sigma^2$. The user should be able to set values for the mean and variance as input arguments.

4.6. Write a function that will generate chi-square random variables with $\nu$ degrees of freedom by generating $\nu$ standard normals, squaring them and then adding them up. This uses the fact that

$$X = Z_1^2 + \ldots + Z_\nu^2$$

is chi-square with $\nu$ degrees of freedom. Generate some random variables and plot in a histogram. The degrees of freedom should be an input argument set by the user.

4.7. An alternative method for generating beta random variables is described in Rubinstein [1981]. Generate two variates $Y_1 = U_1^{1/\alpha}$ and $Y_2 = U_2^{1/\beta}$, where the $U_i$ are from the uniform distribution. If $Y_1 + Y_2 \leq 1$, then

$$X = \frac{Y_1}{Y_1 + Y_2},$$

is from a beta distribution with parameters $\alpha$ and $\beta$. Implement this algorithm.

4.8. Run Example 4.4 and generate 1000 random variables. Determine the number of variates that were rejected and the total number generated to obtain the random sample. What percentage were rejected? How efficient was it?
4.9. Run Example 4.4 and generate 500 random variables. Plot a histogram of the variates. Does it match the probability density function shown in Figure 4.3?

4.10. Implement Example 4.5 in MATLAB. Generate 100 random variables. What is the relative frequency of each value of the random variable 1, ..., 5? Does this match the probability mass function?

4.11. Generate four sets of random variables with \( \nu = 2, 5, 15, 20 \), using the function \texttt{cschirnd}. Create histograms for each sample. How does the shape of the distribution depend on the degrees of freedom \( \nu \)?

4.12. Repeat Example 4.13 for larger sample sizes. Is the agreement better between the observed relative frequencies and the theoretical values?

4.13. Generate 1000 binomial random variables for \( n = 5 \) and \( p = 0.3, 0.5, 0.8 \). In each case, determine the observed relative frequencies and the corresponding theoretical probabilities. How is the agreement between them?

4.14. The MATLAB Statistics Toolbox has a GUI called \texttt{randtool}. This is an interactive demo that generates random variables from distributions that are available in the toolbox. The user can change parameter values and see the results via a histogram. There are options to change the sample size and to output the results. To start the GUI, simply type \texttt{randtool} at the command line. Run the function and experiment with the distributions that are discussed in the text (normal, exponential, gamma, beta, etc.).

4.15. The plot on the right in Figure 4.6 shows a histogram of beta random variables with parameters \( \alpha = \beta = 0.5 \). Construct a similar plot using the information in Example 4.9.