Modelling and Simulation in R

W. John Braun
University of British Columbia

November 22, 2022

Acknowledge chat with Chad Davis about congruences
Ladan worked on the Markov chains chapter
The book was typeset in Latex (Lamport, 1994) using the *knitr* package (Xie, 2018) in R (R Core Team, 2022).

Contents

1

Wildfire, Board Games and Statistics

1.1 Bootstrapping a deterministic fire growth model

Data on the areas and corresponding simulated areas for 99 fires in a subregion of the Muskoka District of Ontario that burned between 1980 and 2010 are plotted in the box plots below. Muskoka is home to a vast number of very expensive properties and is heavily forested but with many lakes and streams fragmenting the landscape from a wildfire perspective.

The actual fires were mostly detected almost immediately, so the time of burning before suppression was much less than a day in most cases. The simulated fires were allowed to burn for 1 hour under the weather conditions that were present on the day that the fire actually burned.

The simulator is called Dionysus and is described in the paper by Han and Braun (2014). It provides an example of bootstrap simulation applied to a deterministic fire growth simulator, in this case, the Prometheus Wildland Fire Growth model (Tymstra et al., 2010).

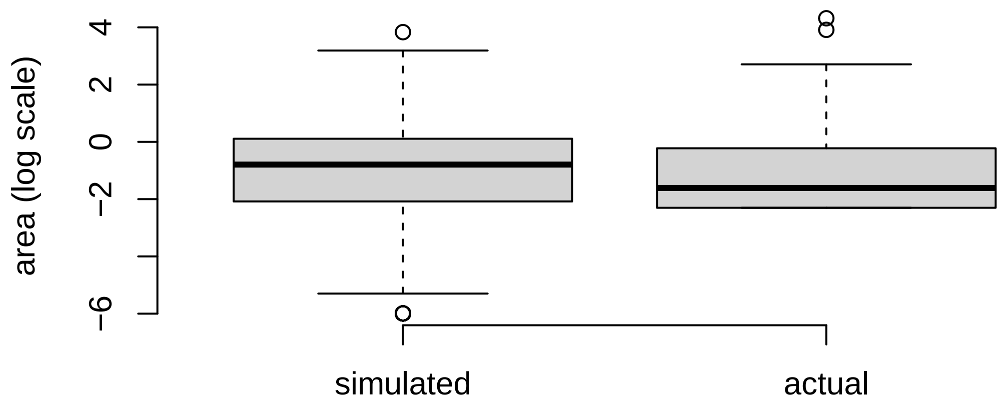


Figure 1.1 shows results for the simulator for one of the fires. The different types of green areas represent different types of fuel (i.e. vegetation) and the blue areas represent bodies of water. The yellow areas represent the simulated burned areas according to the percentiles 0.1, 0.25, 0.5, 0.75, 0.9, read from left to right. For example, the probability that the fire is contained in the yellow region in the left-most panel is 10%, while the probability that it is contained in the yellow region in the right-most panel is 90%.

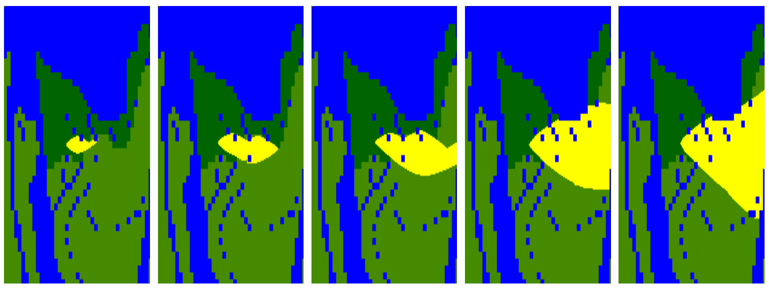
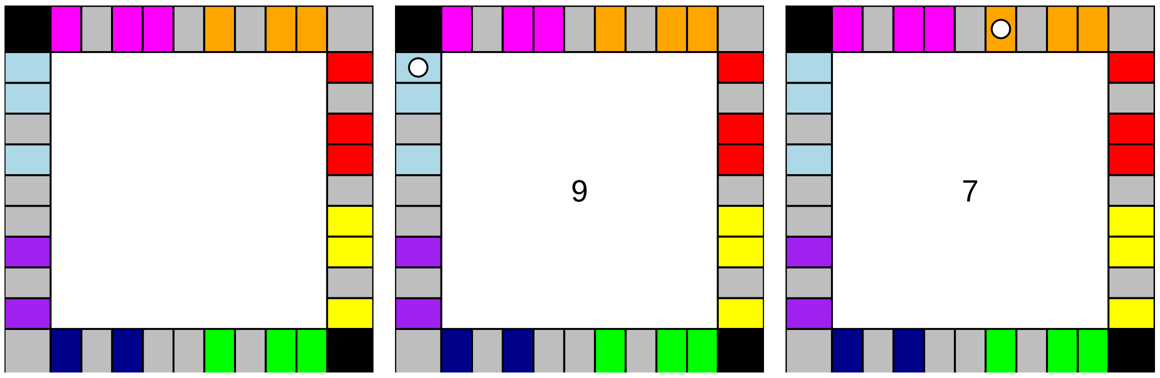


Figure 1.1: Simulated wildfire in the Muskoka region, burning for 1 hour. The yellow region in the first panel corresponds to the amount of fire burned at the 10th percentile; panel two gives the 25th percentile, panel three gives the 50th percentile, panel four gives the 75th percentile, and panel five gives the 90th percentile.

1.1.1 Simulating a popular long-standing board game

Monopoly is a game played by rolling a pair of dice and moving an object (called a token) around colored locations called properties according to the sum of the number of observed dots



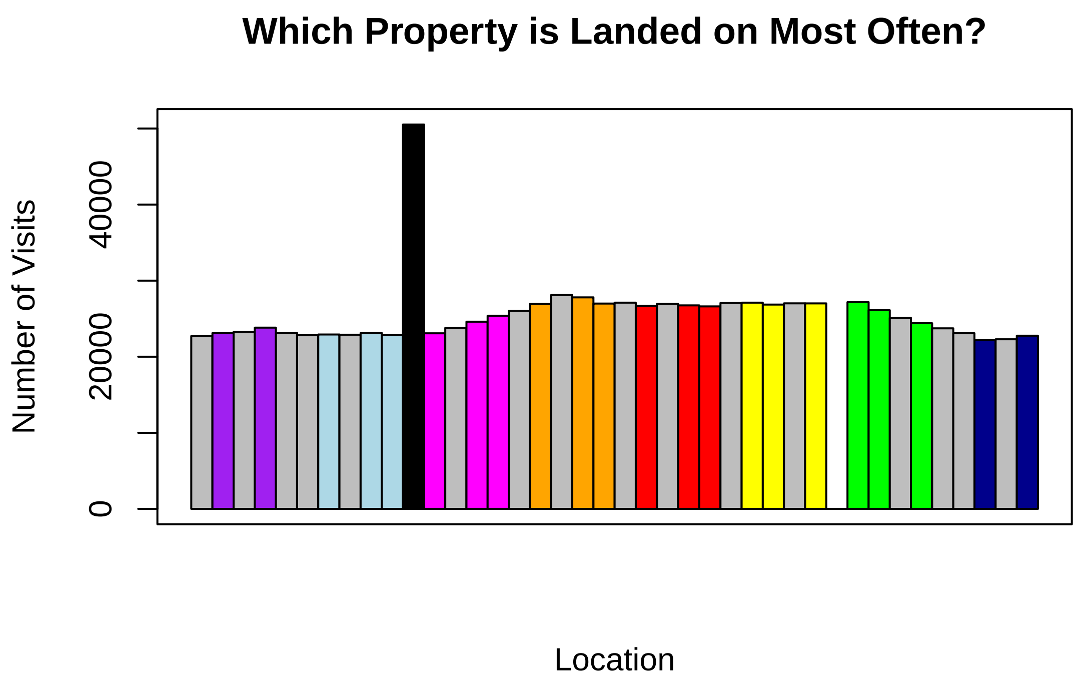
The left panel shows the various color-coded properties of the Monopoly board. The color coding has been chosen to match the colored properties used in the actual game. The grey bars represent non-colored areas of the board such as railroad, utilities, and so on. The black bar represents Jail. The results of the first two dice throws for a single player are shown in the second and third panels. The white circle (the token) represents the player's location on the board. The value at the center represents the sum of the values on the dice.

Having demonstrated a few moves on the Monopoly board, we are now ready to conduct a major simulation to determine which properties are most frequently landed on. The basic code for the simulation is as follows:

```
Nplays <- 1000000
lands <- numeric(Nplays)
lands[1] <- 1                    # The player starts at Go.
for (i in 2:Nplays){
    currentthrow <- sample(1:6, size=2, replace=TRUE)
    lands[i] <- (lands[i-1] + sum(currentthrow) - 1)%%40 + 1
    if (lands[i]==31) lands[i] <- 11   # "Go To Jail" is in position 31
}
```

On a 2 GHz laptop computer, this code takes less than 2 seconds to execute, resulting in the sequence of properties landed on by a single individual in 1000000 moves. The following code produces a histogram whose bars are color-coded according to the property colors.

```
hist(lands, breaks=seq(.5, 40.5, 1), col=propertycolors)
```



Is it surprising that the dark blue properties are not landed on as frequently as the reds and oranges? Why is Jail so popular? Notice the increase in frequency as one proceeds from Jail through the light purple properties towards the orange properties; how is this increase related to the frequencies of the values of pairs of dice?

See `http://rtricks4kids.ok.ubc.ca/test/RTrix/RT4K/Monopoly/` for a simple Monopoly simulator for one player.

1.2 Comparing statistical estimates via simulation

A common problem in statistical research is to compare the quality of different estimators by simulation. In this section, we will compare the sample mean and sample median, using simulated random data.

1.2.1 Background

It is known that the median can sometimes be more appropriate than the mean when measuring central tendency. For symmetric distributions, both the mean and the median are unbiased. For normal data, it is known that the variance of the mean is less than the variance of the median. When there are outliers, the median is often recommended over the mean.

1.2.2 Simulation study

In our study, we

- simulated random data from several different distributions: normal, t_2, t_{10}, and t_{20} at sample sizes $n = 10, 30$ and 100.
- calculated the means and medians for each sample
- created a boxplot of the means and medians for visual comparison
- computed the variances of the means and medians for numerical comparison

Sample Size: 10		
	variance of means	variance of medians
normal	0.10	0.14
t20	0.12	0.15
t10	0.13	0.15
t2	1.70	0.23
Sample Size: 30		
	variance of means	variance of medians
normal	0.04	0.05
t20	0.04	0.05
t10	0.04	0.05
t2	0.70	0.08
Sample Size: 100		
	variance of means	variance of medians
normal	0.01	0.01
t20	0.01	0.02
t10	0.01	0.02
t2	0.11	0.02

Table 1.1: Variances of averages and variances of medians for the samples coming from four different distribution types and three different sample sizes.

1.2.3 Results: tabular variance comparisons

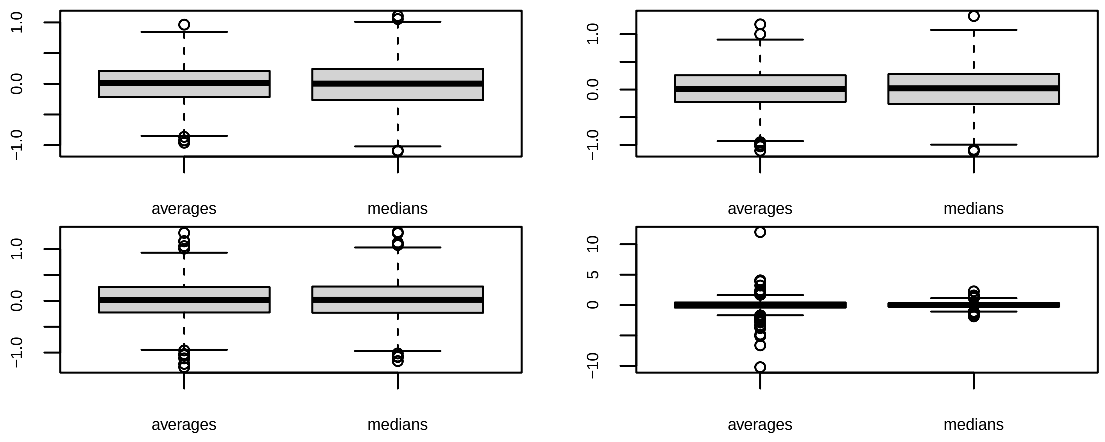


Figure 1.2: Mean versus median comparisons (sample size: 10)

1.2.4 Conclusions

For normal, t_{20}, and t_{10}, the mean has smaller variance than the median, independent of sample size. For t_2 data, the mean has a much larger variance. The variability of the mean and median tends to be similar except for the t_2 case, where the variability of the mean can be quite extreme.

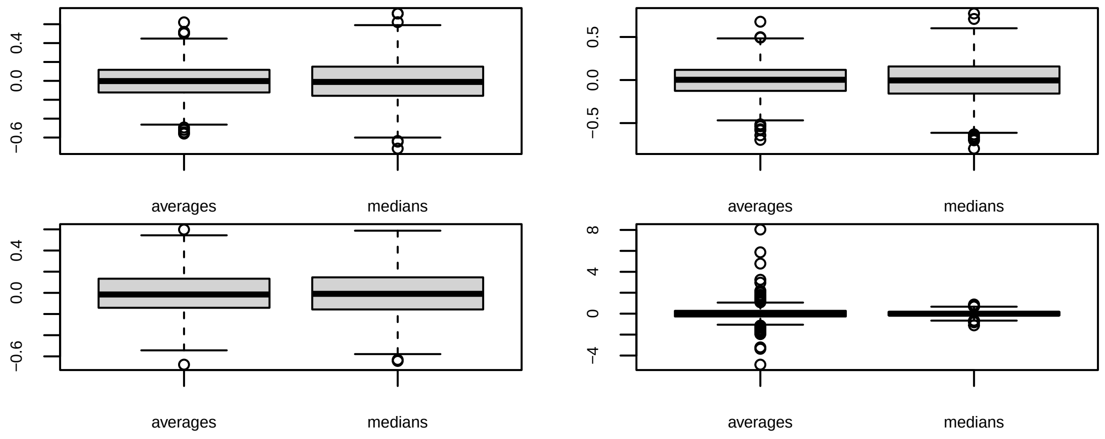


Figure 1.3: Mean versus median comparisons (sample size: 30)

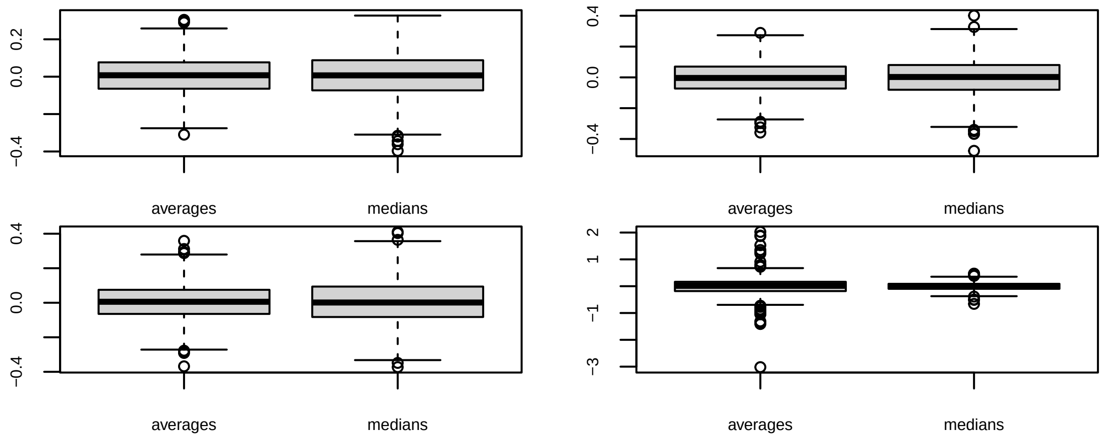


Figure 1.4: Mean versus median comparisons (sample size: 100)

2

Simulation I: Generating and Testing Pseudorandom Numbers

Simulation of realistic scenarios usually requires the incorporation of uncertain or unpredictable elements. This chapter describes how to simulate random numbers on a computer. We will describe deterministic[1] methods of generating values which are then treated as though they are random. Simulated random numbers are called pseudorandom numbers, because they are usually not truly random, but rather, on occasion, a good approximation to truly random numbers.

2.1 Sequential generation of pseudorandom numbers

The most common techniques for constructing pseudorandom variables on a computer are based on sequential generation. That is, previous values are used as inputs to some sufficiently complicated function whose output is the next value in the sequence.

2.1.1 Linearity is highly predictable

If we attempt to generate a sequence by using a linear function of the form

$$x_n = a + bx_{n-1}$$

where a and b are fixed constants, it will be too easy to predict successive members of our sequence. This would define, possibly, the worst of all pseudorandom number generators, since it would be perfectly predictable.

Example 2.1 *Suppose, for* $n = 1, 2, \ldots,$

$$x_n = 3 + 2x_{n-1},$$

and $x_0 = 2$. *Using the above formula with* $n = 1, 2$ *and 3, we obtain*

$$x_1 = 7, x_2 = 17, x_3 = 37.$$

With or without the use of the formula, we can predict that the next numbers would 57, 77, and so on. This kind of predictability is obviously not a good characteristic of a random number simulator.

2.1.2 Nonlinearity can lead to less predictable sequences

To obtain less predictable sequences, we will require a function that will variously lead to an increase or a decrease. Only nonlinear functions have such a property.

Example 2.2 *An example of a nonlinear function is the cosine function. We start with* $x_0 = 2$ *and generate 12 successive values from*

$$x_n = \pi \cos(x_{n-1}).$$

[1]i.e. non-random

```
x <- 2; prnumbers <- numeric(12)
for (n in 1:12) {
    x <- pi*cos(x)
    prnumbers[n] <- x
}
prnumbers

##  [1] -1.3073638  0.8180586  2.1477164 -1.7135662 -0.4470026  2.8329212
##  [7] -2.9931147 -3.1070269 -3.1397161 -3.1415871 -3.1415927 -3.1415927
```

The first few numbers produced by this function are certainly less predictable than the ones generated by the linear map, but eventually, this mapping converges to a single number.

The convergence in the above example occurs because the mapping $x = \pi\cos(x)$ has a fixed point at $x = -\pi$, since $\cos(-\pi) = -1$, and this fixed point is stable, meaning that if x_{n-1} is larger than the fixed point, then $x_n = \pi\cos(x_{n-1})$ will be smaller than x_{n-1}, and if x_{n-1} is smaller than the fixed point, then x_n will be larger than x_{n-1}, and in both cases, x_n will be closer to the fixed point than x_{n-1} was.

The stability of a fixed point is related to the slope of the curve $f(x)$ in a neighbourhood around the fixed point; if the slope is less than 1 in absolute value, the point is stable. This kind of stability is not a useful characteristic of a random number simulator. Instead, we need functions for which any fixed points would be unstable, that is, where the slope at any fixed point is larger than 1 in absolute value.

Example 2.3 *We can increase the frequency of the waveform described by the cosine function increasing the number of possible fixed points in the interval $[-1, 1]$ but also assuring that they are not stable. This mapping is plotted in Figure 2.1, together with the function $f(x) = x$ overlaid, so we can see a large number of fixed points.*

```
curve(cos(30*x), -1, 1)
abline(0, 1)
```

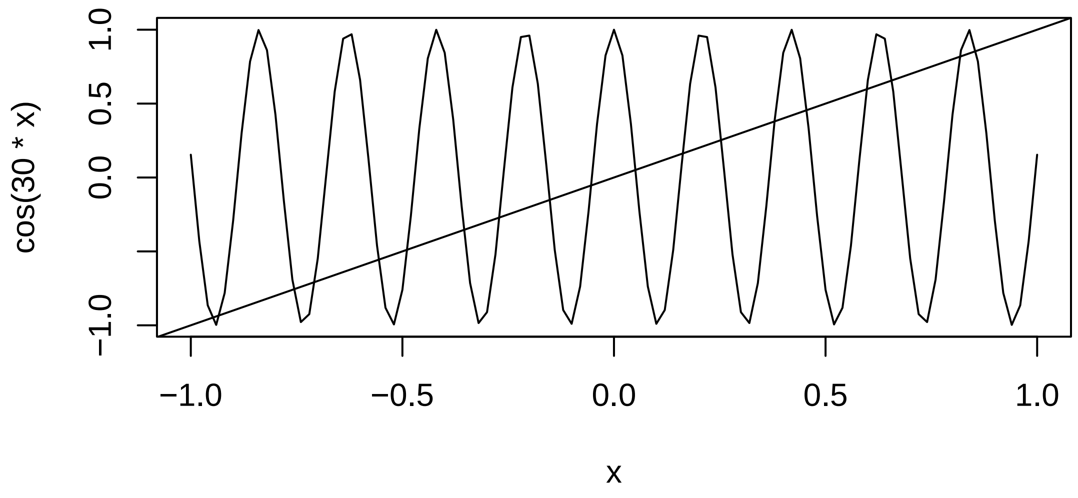


Figure 2.1: A mapping with more, and less stable, fixed points.

Note that the slopes near fixed points (points of intersection between the overlaid line and the curve) are also relatively large, inducing instability.

Generating values from this map, we start with $x_0 = 2$ and calculate 40 successive values from

$$x_n = \cos(30x_{n-1}).$$

```
x <- 2; prnumbers <- numeric(40)
for (n in 1:40) {
    x <- cos(30*x)
    prnumbers[n] <- x
}
ts.plot(prnumbers)
```

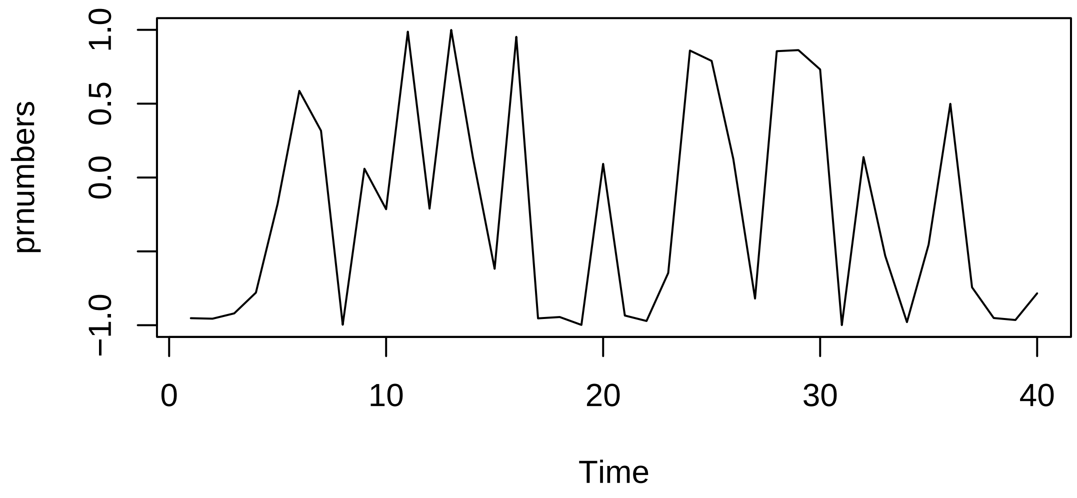


Figure 2.2: Trace plot of first 40 numbers sequentially generated from the cosine map, $\cos(30x)$.

The numbers produced by this function are certainly less predictable than before, as can be seen in the trace plot pictured in Figure 2.2.

Functions with jumps can also provide mappings which are very unpredictable.

Example 2.4 *Consider the function* $f(x) = 32678x \mod 33271$*:*

```
modfun <- function(x) {
    (32678*x)%%33271
}
```

Its graph is plotted in Figure 2.3.

```
curve(modfun, 0, 30000)
abline(0,1)
```

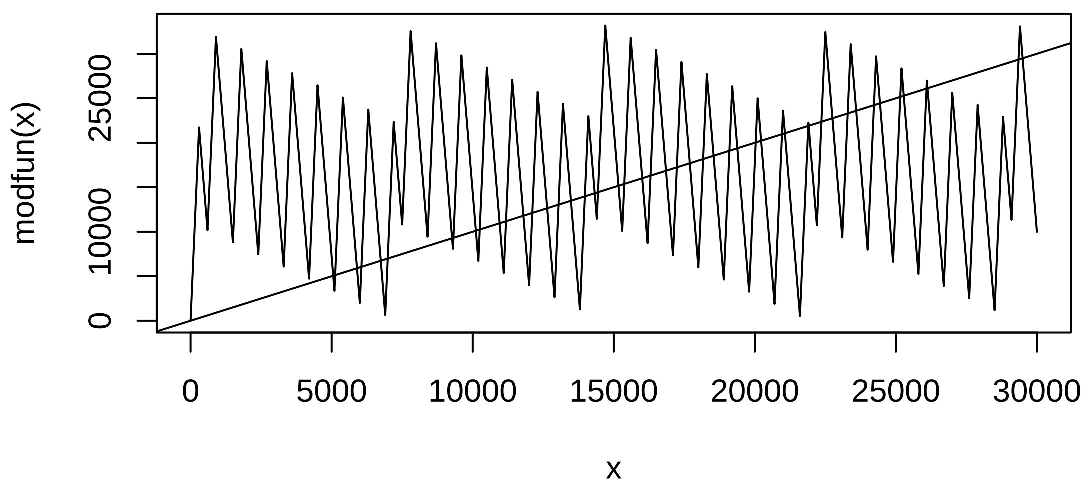


Figure 2.3: An example of a nonlinear function with jumps.

```
x <- 2; prnumbers <- numeric(40)
for (n in 1:40) {
    x <- modfun(x)
    prnumbers[n] <- x
}
ts.plot(prnumbers)
```

The first 40 pseudorandom numbers produced by this function are traced in Figure 2.4, showing a much less predictable pattern than any we have seen so far.

2.1.3 Multiplicative congruential pseudorandom number generators

The example of the preceding section is suggestive of one of the simplest methods for simulating independent uniform random variables on the interval [0,1]: the multiplicative congruential random number generator.[2] The generator produces a sequence of pseudorandom numbers, $u_0, u_1, u_2, \ldots,$ which can appear like independent uniform random variables on the interval [0,1].

Let m be a large integer, and let a be another integer which is smaller than m. The sequence x_n is then generated, for $n = 1, 2, \ldots$,

$$x_n \quad = \quad ax_{n-1} \bmod m$$

[2]Some other generators will be studied in a later section of this chapter.

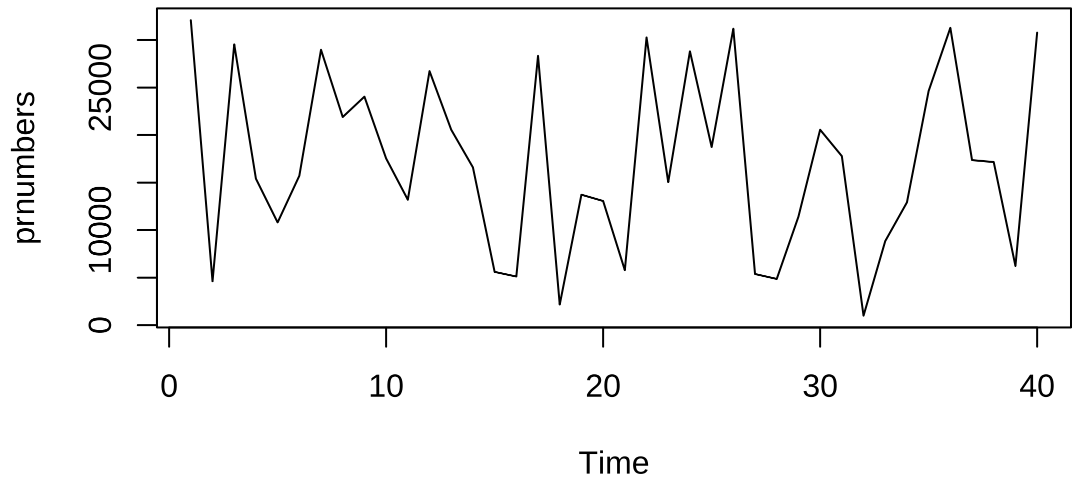


Figure 2.4: Trace plot of the first 40 numbers coming from the nonlinear function with jumps.

starting with the seed, x_0, which is taken as some integer value between 1 and m. To ensure that the x_n values lie in the interval $[0, 1]$, we divide each by m:

$$u_n = x_n/m.$$

The quality of the generator depends on the values of a and m. Small values of m are not recommended, since there can be no more than m distinct values generated according to the method.

In fact, if a and m are not chosen well, it is possible for the sequence of values to repeat or exhibit cycling before the full set of m values is obtained.

Example 2.5 *The code below produces 6 pseudorandom numbers based on the multiplicative congruential generator:*

$$x_n = 7x_{n-1} \textit{ mod } 32$$

$$u_n = x_n/32$$

with initial seed $x_0 = 2$.

```
random.number <- numeric(6)   # the output
                             # will be stored here
random.seed <- 2
for (j in 1:6) {
     random.seed <- (7 * random.seed) %% 32
     random.number[j] <- random.seed/32
 }
```

The first 6 values in the sequence are

```
random.number[1:6]

## [1] 0.4375 0.0625 0.4375 0.0625 0.4375 0.0625
```

Note that only 2 distinct values were obtained, since $7 \times 2 = 14$ *and* $14 \times 7 = 98$*, the latter being 2 modulo 32. The cycle length is 2, which is much less than the 32, we might have hoped for.*

The next example gives a generator with somewhat better behaviour.

Example 2.6 *The code below produces 30268 pseudorandom numbers based on the multiplicative congruential generator:*

$$x_n = 171x_{n-1} \text{ mod } 30269$$

$$u_n = x_n/30269$$

with initial seed $x_0 = 27218$.

```
random.number <- numeric(30268)  # the output
                                 # will be stored here
random.seed <- 27218
for (j in 1:30268) {
     random.seed <- (171 * random.seed) %% 30269
     random.number[j] <- random.seed/30269
 }
```

The results, stored in the vector `random.number`*, are in the range between 0 and 1. These are the pseudo-random numbers,* $u_1, u_2, \ldots, u_{30268}$.

```
random.number[1:50]

##  [1] 0.76385080 0.61848756 0.76137302 0.19478675 0.30853348
##  [6] 0.75922561 0.82757937 0.51607255 0.24840596 0.47741914
## [11] 0.63867323 0.21312234 0.44391952 0.91023820 0.65073177
## [16] 0.27513297 0.04773861 0.16330239 0.92470845 0.12514454
## [21] 0.39971588 0.35141564 0.09207440 0.74472232 0.34751726
## [26] 0.42545178 0.75225478 0.63556774 0.68208398 0.63636063
## [31] 0.81766824 0.82126929 0.43704780 0.73517460 0.71485678
## [36] 0.24051009 0.12722587 0.75562457 0.21180085 0.21794575
## [41] 0.26872378 0.95176583 0.75195745 0.58472364 0.98774324
## [46] 0.90409330 0.59995375 0.59209092 0.24754700 0.33053619
```

```
length(unique(random.number))

## [1] 30268
```

The last calculation shows that this generator did not cycle before all possible numbers were computed.

2.1.4 A multiplicative congruential generator function

The following function will produce `n` simulated random numbers on the interval $[0, 1]$, using a multiplicative congruential generator:

```
rng <- function(n, a=171, m=30269, seed=1) {
    x <- numeric(min(m-1,n))
    x[1] <- seed
    for (i in 1:min(m-1,n)){
        y <- x[i]
        x[i+1] <- (a*y)%%m
    }
    x[2:(n+1)]/m
}

rng(5)  # simple example of use of rng

## [1] 0.005649344 0.966037861 0.192474148 0.913079388 0.136575374
```

2.2 Are the simulated numbers adequate for the problem at hand?

Testing of pseudorandom number generators is a critical topic and some general guidance is provided in the section following this one. However, it must be kept in mind that the nature of the problem requiring random numbers is an important consideration. Generally, simpler problems will make fewer demands on the quality of the numbers generated, while complex problems such as those arising in theoretical physics or genomics may be too demanding for even the best of the currently available generators.

The paper by O'Neill (2014) summarizes the modern goals of random number generators succinctly. Desirable features are long cycle length, output range, uniform distribution, lack of predictability, repeatability with a given seed, robustness to adversarial attacks (if the seed is unknown, inverting the sequence to find the seed should be difficult), speed, and memory considerations.

In this section, we consider a very simple problem which illustrates some of the issues. Consider the generator based on

$$x_n = 7x_{n-1}\text{mod}17.$$

Using $x_0 = 1$, we obtain the following values in the sequence before it begins to cycle:

```
rng(16, 7, 17)*17

##  [1]  7 15  3  4 11  9 12 16 10  2 14 13  6  8  5  1
```

2.2.1 *Tossing two fair coins*

The problem we want to apply this generator to is that of simulating the tossing of two fair coins. We will use the rule that a 'Head' (H) is generated whenever the generated value is less than 9, and otherwise a 'Tail' (T) is generated. Thus, we could use the above sequence to generate the following pattern of heads and tails:

```
##  [1] "H" "T" "H" "H" "T" "T" "T" "T" "T" "H" "T" "T" "H" "H"
## [15] "H" "H"
```

Note that we only require a single consecutive pair of coin tosses, not the entire sequence. Thus, if we request only 2 values from the generator and seed it with the value 1, we get an H-T outcome, while if we seed with the value 7, we get a T-H outcome, and so on. The outcomes for each choice of seed are given in Table 2.1.

Table 2.2 displays the frequency of occurrence for each possible outcome. The table shows that if one chooses the seed randomly from the set $\{1, 2, \ldots, 16\}$, a pair of heads will occur with probability 5/16 as is the case for a pair of tails. On the other hand, H-T and T-H pairs occur with probability 3/16 each. Thus, the generator will give a biased result for this simple problem, since we should be expecting to see any of these four outcomes with exactly the same probability.

2.2.2 *Obtaining the correct solution by restricting the choice of seed*

Notice that if one seeds the generator with one of $\{4, 11, 9, 12, 2\}$, a T-T pair will result while seeding with one of $\{13, 6, 8, 5, 15\}$ will yield an H-H outcome. Thus, we can reduce the probability of both of these outcomes by

seed	outcome
1	H T
7	T H
15	H H
3	H T
4	T T
11	T T
9	T T
12	T T
16	T H
10	H T
2	T T
14	T H
13	H H
6	H H
8	H H
5	H H

Table 2.1: Simulated pairs of coin tosses for given seeds.

H H	5
H T	3
T H	3
T T	5

Table 2.2: Observed frequencies of successive coin toss outcomes

disallowing certain seeds. Removing two from each set will reduce number of possible T-T and H-H outcomes to three, matching the H-T and T-H frequencies. Removing seeds at the extremes (i.e. either too large or too small) is a simple general strategy that often leads to improved performance. Thus, we could disqualify seeds 2, 4, 13 and 15. Choosing any other seed will result in a pair of coin toss outcomes that exactly follows the required probability distribution.

2.2.3 *Tossing three fair coins*

Table 2.4 displays the frequency of occurrence for each possible outcome. The table shows that if one chooses the seed randomly from the set $\{1, 2, \dots, 16\}$, the different outcomes are not equally likely. As in the case of two tosses, we can ensure that each possible triple has equal chance of occurrence by disqualifying certain seed values. The only way that equally likely outcomes are assured is if only one occurrence of each outcome is possible. By restricting the possible seeds to the set $\mathbf{1}, 3, 6, 7, 9, 12, 15, 16\}$, the generator will be capable of producing a set of three independent coin tosses.

2.2.4 *Tossing four fair coins*

Table 2.6 displays the frequency of occurrence for each possible outcome. The table shows that if one chooses the seed randomly from the set $\{1, 2, \dots, 16\}$, the different outcomes are not equally likely, and worse yet, some of the 16 expected outcomes are not possible. This time, we cannot ensure that each possible quadruple has an equal chance of occurrence by disqualifying certain seed values. This generator will fail to work on this very simple problem.

seed	outcome
1	H T H
7	T H H
15	H H T
3	H T T
4	T T T
11	T T T
9	T T T
12	T T H
16	T H T
10	H T T
2	T T H
14	T H H
13	H H H
6	H H H
8	H H H
5	H H T

Table 2.3: Simulated triples of coin tosses for given seeds.

H H H	3
H H T	2
H T H	1
H T T	2
T H H	2
T H T	1
T T H	2
T T T	3

Table 2.4: Observed frequencies of successive coin toss outcomes

2.3 Basic checks on RNG quality

There are two essential requirements for a good sequence of pseudorandom numbers. First, the distribution of numbers should be uniform on the interval $[0, 1]$. Second, the successive values should be sequentially independent. That is, it should not be possible to predict values in the sequence from other values.

A number of tests have been developed to investigate the quality of pseudorandom number generators.

2.3.1 Histogram

The most straightforward check on the uniform distribution assumption is to plot a histogram of a sample coming from a generator. A perfect-looking rectangular distribution is often a symptom of too much order in the generator, while gaps in the histogram are also an indicator that the generator is failing to meet the uniform distribution requirement.

Example 2.7 *Sequences of pseudorandom numbers are stored in* `x1`*,* `x2` *and* `x3`*:*

```
x1 <- rng(1000, a = 32377, m = 32378); x2 <- rng(1000, a = 41, m = 2000)
    x3 <- runif(1000)
par(mfrow=c(1,3)); hist(x1); hist(x2); hist(x3)
```

Figure 2.5 displays the histograms of the 3 sequences. `x1` *fails;* `x2` *and* `x3` *both appear to be uniform, though the middle histogram is actually too perfect to be believable.*

seed	outcome
1	H T H H
7	T H H T
15	H H T T
3	H T T T
4	T T T T
11	T T T T
9	T T T H
12	T T H T
16	T H T T
10	H T T H
2	T T H H
14	T H H H
13	H H H H
6	H H H H
8	H H H T
5	H H T H

Table 2.5: Simulated quadruples of coin tosses for given seeds.

H H H H	2
H H H T	1
H H T H	1
H H T T	1
H T H H	1
H T T H	1
H T T T	1
T H H H	1
T H H T	1
T H T T	1
T T H H	1
T T H T	1
T T T H	1
T T T T	2

Table 2.6: Observed frequencies of successive coin toss outcomes

Sometimes the use of a chi-squared goodness-of-fit test is advocated, but this would usually be a waste of effort, since a glance at the histogram is all that is really needed to check for uniformity. Furthermore, the more important and difficult issue is related to the independence of the sequence of values. More care is required to assess this aspect of a generator.

2.3.2 Visualizing RNG output with a lag plot

We can get an idea of whether we are generating independent random numbers by plotting successive pairs of numbers - a lag 1 plot. Specific patterns indicate that the generator is not producing independent numbers. If you see points lining up, your generator is failing.

Example 2.8 *In the first example, we see points lying on clearly separated lines in the lag plot displayed in Figure 2.6. This generator is very poor.*

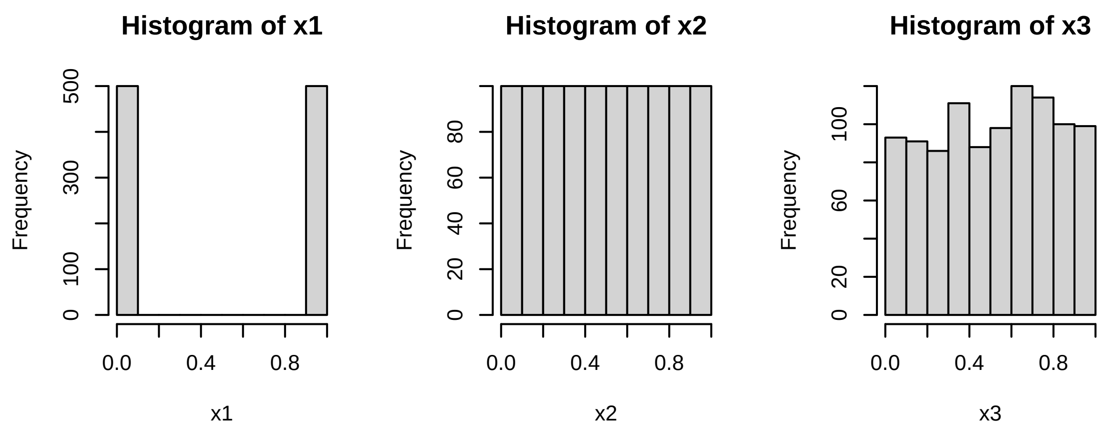


Figure 2.5: Histograms of three sets of pseudorandom numbers.

```
x <- rng(100,a=15, m=511,seed=12)
lag.plot(x, do.lines=FALSE)
```

By changing the a parameter, we get the result plotted in Figure 2.7.

```
x <- rng(100,a=31, m=511,seed=12)
lag.plot(x, do.lines=FALSE)
```

Plotted points appear to be more random in this second example.

In our third example, we take a much larger value of m, together with a value of a which is in the order of the square root of m. This kind of combination can lead to better behaviour, and the lag plot picture in Figure 2.8 confirms this.

```
x <- rng(100, a=171, m=30269, seed=12)
lag.plot(x, do.lines=FALSE)
```

2.3.3 Autocorrelation

The autocorrelation function, ACF, numerically summarizes what can be observed graphically on a lag plot. The autocorrelation at a particular lag, say m, is the correlation between the $n + m$th value and the nth value, or more precisely, the correlation between the vector $(x_1, \ldots, x_{n-m})$ and the vector $(x_m, x_{m+1}, \ldots, x_n)$.

Autocorrelations, like correlations, can take values between -1 and 1, where positive values are indicative of the plotted points scattering about a line of positive slope, and negative values are indicative of the plotted points scattering about a line of negative slope.

Example 2.9 *Figure 2.9 shows the first 6 lag plots for one of the sequences generated earlier. In other words, the figures show plots of the $n + 1$st elements versus the nth, $n + 2$st versus the nth, $n + 3$st versus the nth and so on.*

```
lag.plot(x2, lag=6)
```

At lags 1, 2, 3, 4 and 6, it would be hard to predict the current value of x2, but the lag 5 plot shows that the current value of x2 depends a lot on the value 5 time units earlier.

The autocorrelations for the first 5 lags are:

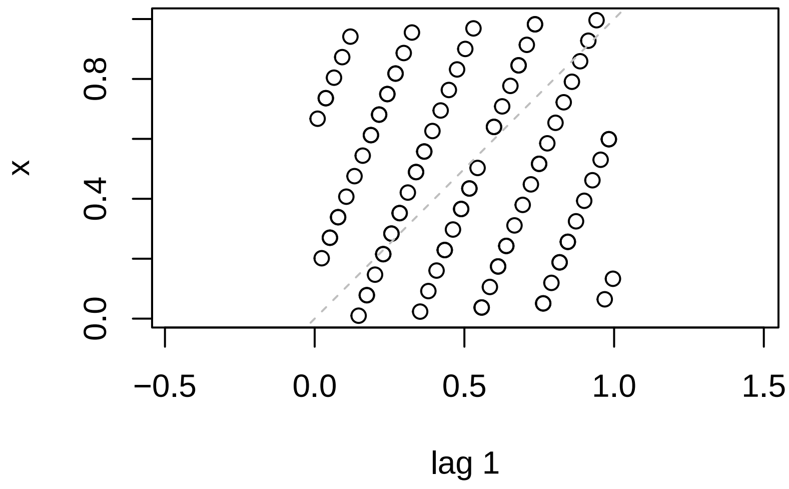


Figure 2.6: Lag plot for a pseudorandom number generator with bad properties.

```
acf(x2)$acf[2:6]   #
```

```
## [1] -0.0328  0.0049 -0.1090 -0.1097  0.4575
```

The values of this autocorrelation function are plotted in Figure 2.10. Note the size of the 5th one. This says that every 5th value of the sequence is dependent.

The autocorrelations for the first 5 lags for values coming from `runif()` *are:*

```
acf(x3)$acf[2:6]   #
```

```
## [1] -0.012205 -0.001414 -0.012534  0.000379  0.003676
```

Again, the autocorrelation function is plotted in Figure 2.11. All ACF values checked are small. This sequence passes this test.

Note that a severe deficiency of checking the autocorrelations is that they only detect linear forms of dependence and can miss nonlinear dependencies. Thus, they really only serve as a quick way to check many lags at once; the lag plots have the advantage of highlighting nonlinear dependencies, if they are there. Both methods will fail to show more complex dependence structures, where, for example, values can be predicted nonlinearly by a combination of earlier values in the sequence.

Example 2.10 *In the 1960's, the RANDU pseudorandom number generator was very popular. It was a multiplicative congruential generator with $a = 65539$ and $m = 2^{31}$.*

We can generate numbers from this sequence using

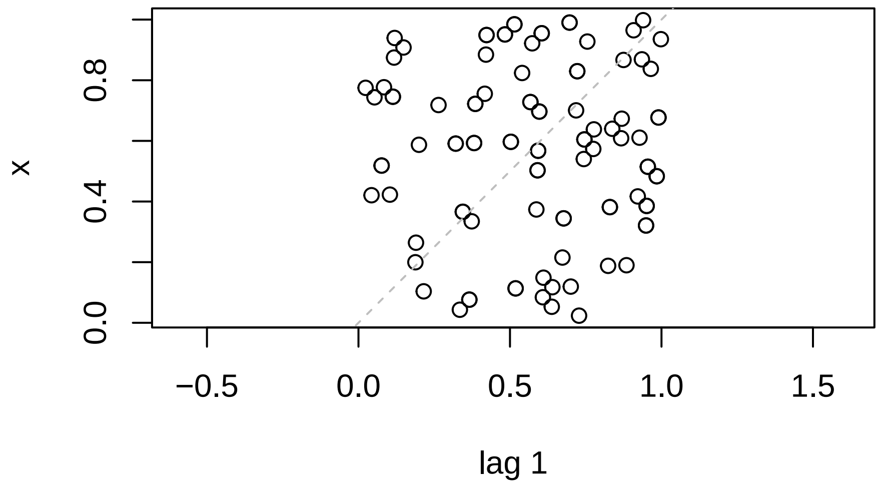


Figure 2.7: Lag plot for a pseudorandom number generator with small m but relatively good properties for very short sequences.

```
n <- 5000
RANDU <- numeric(n)
x <- 123
for (i in 1:n) {
  x <- (65539*x) %% (2^31)
  RANDU[i] <- x / (2^31)
}
```

```
par(mfrow=c(1,2))
hist(RANDU)
acf(RANDU)
```

According to these simple checks, there does not appear to be a problem, and this is a reason for its temporary popularity. However, upon more rigorous checking, a serious deficiency was discovered, when one plots a 3-dimensional version of the lag plot. In other words, plot u_{n+2} against u_{n+1} and u_n. The following code sets up the vectors that make up the 3d lag plot:

```
A <- diag(rep(1,n-1)); A <- rbind(rep(0, n-1), A)
A <- cbind(A, rep(0, n))
lagvectors <- matrix(RANDU, nrow=n)
m <- 2
for (j in 1:m) {
    lagvectors <- cbind(lagvectors, A%*%lagvectors[,j])
}
lagvectors <- data.frame(lagvectors)
names(lagvectors) <- c(paste("x", 1:(m+1), sep=""))
```

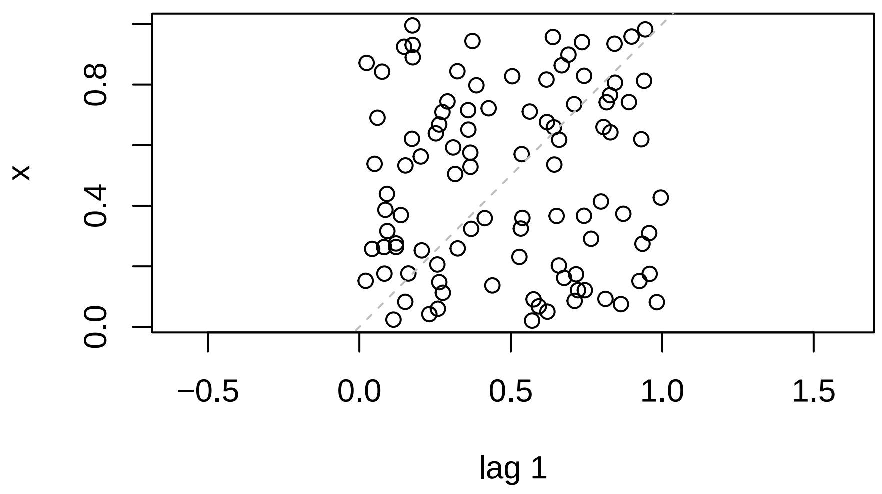


Figure 2.8: Lag plot for a pseudorandom number generator with moderate m and relatively good properties for short sequences.

```
lagvectors <- lagvectors[-(1:m),]
```

Here, we have set up an $n \times n$ matrix A for which Ax yields a vector of length n whose first element is 0 and whose remaining elements are $x_1, \ldots, x_{n-1}$. Thus, A^2x is a vector whose first two elements are 0's and whose remaining elements are $x_1, \ldots, x_{n-2}$. The vectors $x_1 = x, x_2 = Ax, x_3 = A^2x$ are the basis of the 3-dimensional lag plot pictured, from two different orientations, in Figure 2.13. Here, we have used the `scatterplot3d` *function in the scatterplot3d package (Ligges and Mächler, (2003).*

```
library(scatterplot3d)
par(mfrow=c(1,2))
scatterplot3d(lagvectors$x3,lagvectors$x2, lagvectors$x1,
    xlab="x3", ylab="x2", zlab="x1", cex.symbols=.3)
scatterplot3d(lagvectors$x3,lagvectors$x2, lagvectors$x1,
    xlab="x3", ylab="x2", zlab="x1", angle=150, cex.symbols=.3)
```

The change in perspective is important, since most views of the data do not reveal anything other than an apparent random scatter of points - under the given 2-dimensional projection. The perspective taken in the right panel of Figure 2.13 reveals the difficulty with RANDU: it only produces u_n, u_{n+1}, u_{n+2} triples lying on a small number of planes in 3-dimensional space.

Better generators provide successive triples that do a better job of filling 3-dimensional space. In fact, one also desires a generator that will provide successive quadruples that leave only small gaps in 4-dimensional space, and more generally, successive m-tuples that leave only small gaps in m-dimensional space. The spectral test, which lies beyond the scope of this book, is designed to find these deficiencies at a given number of dimensions.

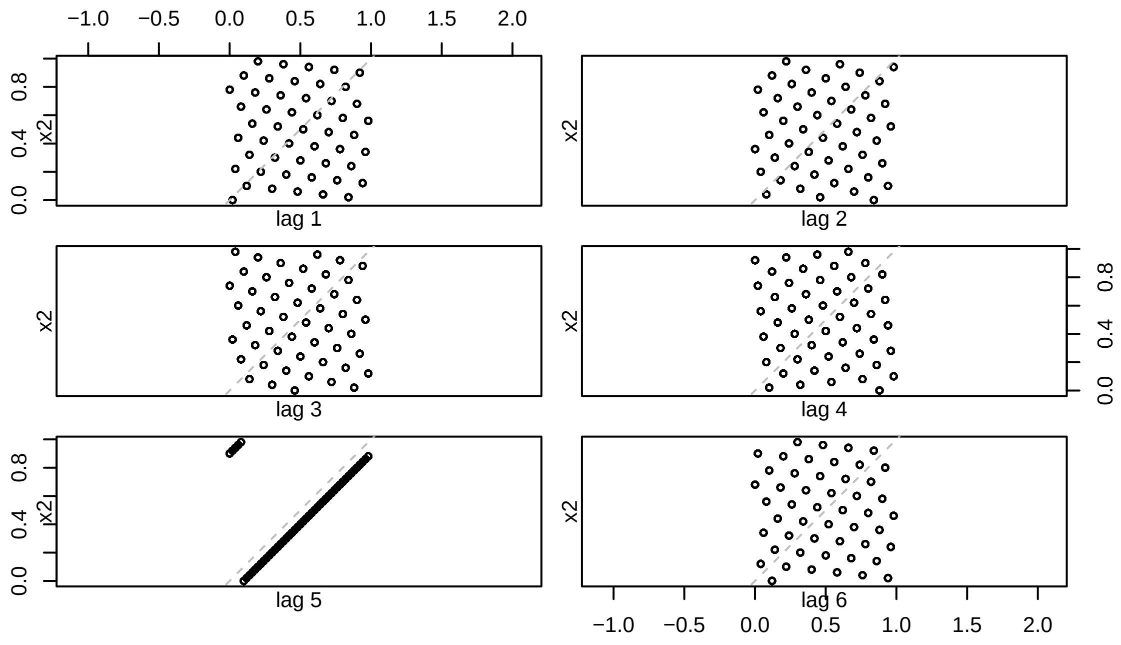


Figure 2.9: Lag plots for the first 6 lags.

2.4 Random forest testing of a pseudorandom number generator

The `randomForest` function in the *randomForest* package (Liaw and Wiener, 2002) can be used to set up a quick and simple approximation to the spectral test. The essential idea behind this test is to set up a flexible prediction model for successive elements of a sequence generated by a pseudorandom number generator, given m previous values. If the predictions are consistently inaccurate, the generator can be judged adequate; when the predictive model is sometimes successful, the generator should be judged a failure.

A quick review of the nature of regression trees and their extension as random forests is necessary before we describe an implementation of the random forest testing approach for pseudorandom numbers.

2.4.1 *Regression trees and random forests*

A regression tree is a flexible or nonparametric approach to predictive modelling which is particularly useful when there are a relatively large number of possible predictors and where the nature of the relationship between the response and the predictors is very much unknown and not likely linear. The *rpart* package (Therneau and Atkinson, 2018) implements a recursive partitioning technique which can produce both classification trees and regression trees. A classification tree is useful in the case where the response variable is categorical, for example, binary. Classification trees offer a flexible alternative to logistic regression. Regression trees offer a flexible alternative to multiple regression.

Example 2.11 *Consider the data in* `table.b3` *of the MPV package (Braun and MacQueen, 2019). This data set concerns gas mileage,* `y`*, for a number of cars, together with information on 11 other variables. In particular, we note that* `x10` *represents weight and* `x2` *represents horsepower. We can fit the default regression tree to these data using*

```
library(rpart); library(MPV)
mpg.tree <- rpart(y ~ ., data = table.b3)
```

The tree, plotted in Figure 2.14, is the result of executing the following code.

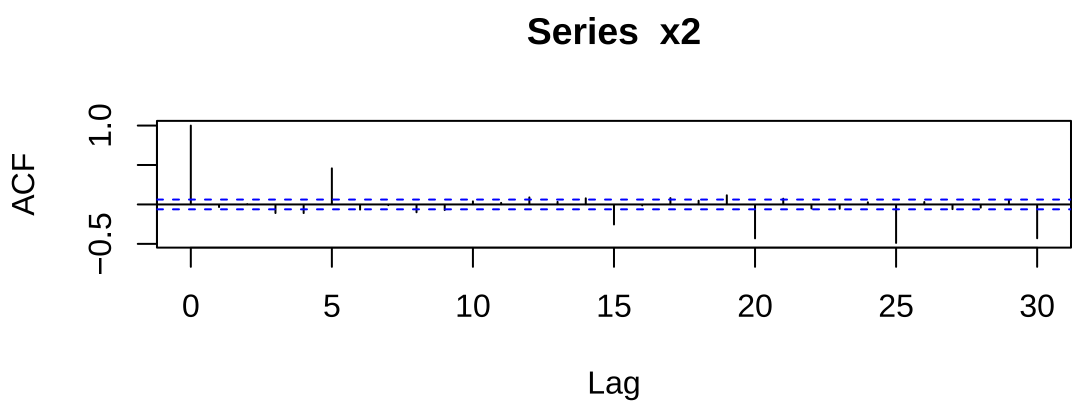


Figure 2.10: Autocorrelations for the x2 sequence.

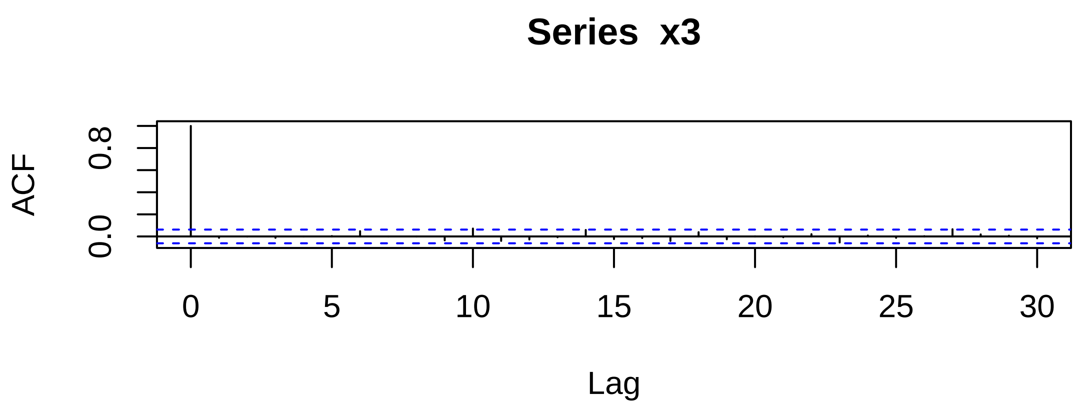


Figure 2.11: Autocorrelations for the x3 sequence.

```
par(xpd = TRUE)
plot(mpg.tree, compress=TRUE)
text(mpg.tree, cex=.5, use.n = TRUE)
```

The tree indicates how the partitioning or splitting of the predictor space was undertaken. For weights (`x10`*) less than 2678 pounds, the gas mileage was predicted to be 29.76 miles per gallon. This split would have been chosen to minimize the prediction error. Then an additional split was made, for the data set where weights which are at least 2678 pounds. In this case, when the horsepower (*`x2`*) is less than 144, the gas mileage is predicted to be 20.3, and the prediction is 15.72 when the horsepower is at least 144. Again, this part of the predictor space was partitioned in order to minimize prediction error. The splitting process was terminated, based on a trade-off between predictive precision and model complexity. Models that have too many splits or branches tend to over-fit the data.*

As the name implies, a random forest is a random collection of trees. The trees are essentially constructed from random samples, taken with replacement, from the original sample of observations. This is an example of a technique called bootstrapping. By averaging the predictions over the entire set of trees, improved stability can

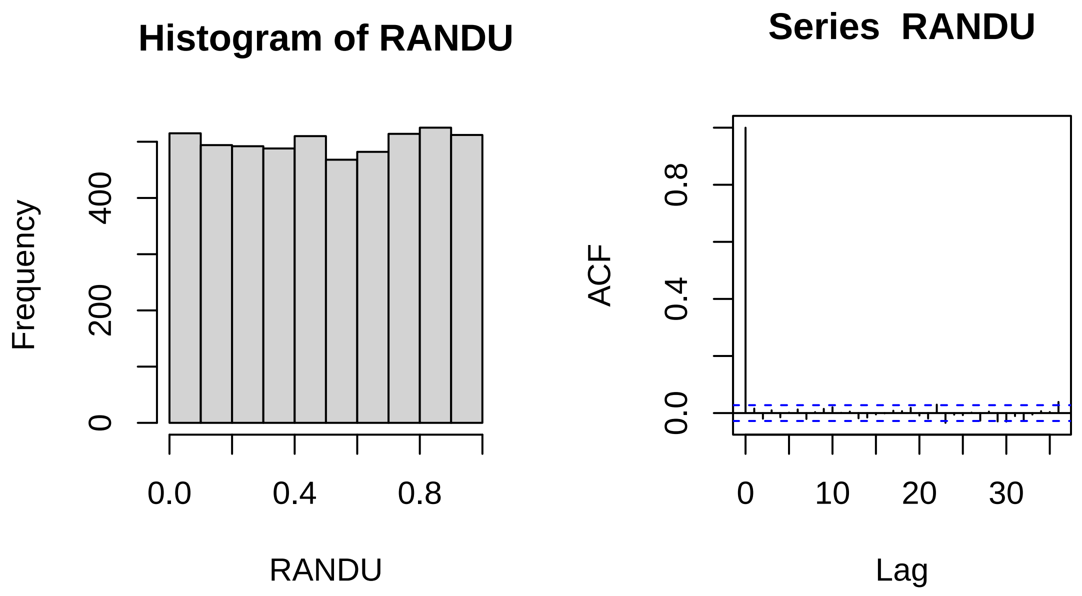


Figure 2.12: Histogram and ACF plots for a sequence of RANDU numbers.

often be achieved.

Example 2.12 *We can apply the random forest approach to the car gas mileage data as follows:*

```
library(randomForest)
mpg.rf <- randomForest(y ~ ., data = table.b3[,-4])
```

Figure 2.15 displays a plot of the actual gas mileages against predictions for both the random forest predictions and the tree predictions. Whether the random forest predictions are better is not necessarily clear. What is clear is that the predictions are somewhat finer grained and somewhat more flexible.

```
par(mfrow=c(1,2), mar=c(4, 4, 1, 1))
plot(table.b3$y ~ predict(mpg.rf), ylab="Actual mpg",
   xlab="Predicted mpg")
plot(table.b3$y ~ predict(mpg.tree), ylab="Actual.mpg",
   xlab ="Predicted mpg")
```

2.4.2 Testing pseudorandom numbers with random forest prediction

The function `rftest()` can be used to carry out the test for a given pseudorandom number sequence, coming from the generator to be tested. Typically, as in the spectral test, one supplies a sequence of m values which represent the dimensionality of the space to be "filled" by the successive m-tuples of sequence values. The function constructs the m vectors as in the RANDU example of Section 2.3.3, and the random forest is then used to set up a predictive model for values of x_{n+m}, given $x_{n+m-1}, x_{n+m-2}, \ldots, x_n$. The fitting is actual done on one-half of the data, the so-called training set. The remaining half of the data, the so-called test set, is plugged into the fitted model to obtain predictions. The actual values of x_{n+m} are plotted against the predictions, first using the training set – internal validation and then using the test set – external validation. In both cases, a scatter plot of the

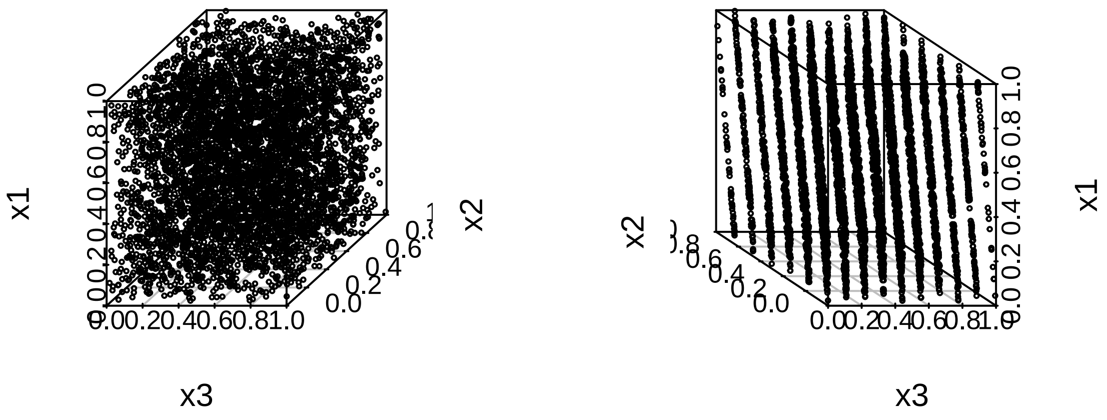


Figure 2.13: 3-dimensional lag plot of a sample from the RANDU simulator, from two different perspectives.

actual values against the predicted values is obtained, with a least-squares line overlaid. A line with positive slope, particularly on the second plot, is an indicator of failure for the generator. One would not expect a line with a substantial negative slope, since the poor predictivity from the random forest should only yield random predictions and not predictions that are negatively correlated with the actual values. Thus, a line with non-positive slope should be interpreted as a success for the generator. This is coded up in the function `rftest` which is available in the MPV package (Braun and MacQueen, 2019).

```
library(MPV) # The function rftest is in the MPV package
```

Example 2.13 *We start with the cosine sequence discussed in Section 2.1.2 to show why such a generator is not in practical use. The results of the random forest test are pictured in Figure 2.16 using $n = 500$ and the default of $m = 5$ dimensions.*

```
for (n in 1:500) {
    x <- cos(30*x)
    prnumbers[n] <- x
}
rftest(prnumbers)
```

Observe that in both plots, the least-squares line has a substantial positive slope. The random forest model is making excellent predictions of x_{n+5} based on the previous 5 observations. This cosine mapping would not be useful in producing good approximations to random numbers.

In practice, we should use as large a sample as is practical, and we should look for trouble over a sequence of m values, starting with 1. In the case of the spectral test, one does not normally go beyond 8 dimensions, but the random forest approximation can be run at higher dimensions without much difficulty.

Example 2.14 *We will check the quality of the default generator in R, using the random forest test, using $n = 5000$, and $m = 1$ and $m = 10$. It is an easy exercise to try intermediate values of m.*

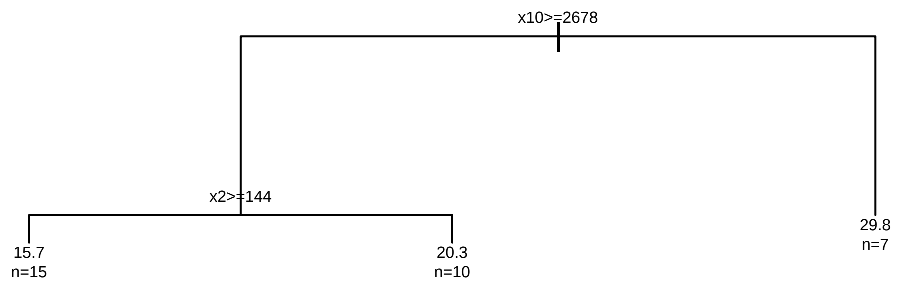


Figure 2.14: Default regression tree for car gas mileage data.

```
u <- runif(5000)
```

Execution of the following scripts yields the plots displayed in Figures 2.17 and 2.18. In both cases, we see that the least-squares lines are effectively constant; in some cases, there is the appearance of a slightly negative slope, which we have discussed earlier as an indication that the random forest is really not able to deliver a predictive model. These plots confirm that, at least for small simulation problems, the default simulator in R is going to work well. Larger values of n should be considered before making conclusions about larger scale simulations.

```
rftest(u, m = 1)
```

```
rftest(u, m = 10)
```

We saw earlier that the RANDU generator has a serious deficiency. Can the random forest test detect this problem?

Example 2.15 *Since the issue for RANDU occurs when $m = 2$, we will apply the random forest test using this value of m.*

```
rftest(RANDU,  m = 2)
```

Figure 2.19 displays the results. As expected, the fitted least-squares line relating the actual sequence values with the random forest predictions has an obviously positive slope, indicating that the random forest is able to find a relatively good predictive model for values in this sequence, based on the preceding two values in the sequence. This result is consistent with the earlier analysis that indicates that the RANDU generator will not produce good unpredictable numbers.

2.5 The linear congruential method

The linear congruential generator is an elementary extension of the multiplicative congruential generator. A linear congruential sequence of random numbers is generated using

$$x_{n+1} = (ax_n + c) \text{ mod } m.$$

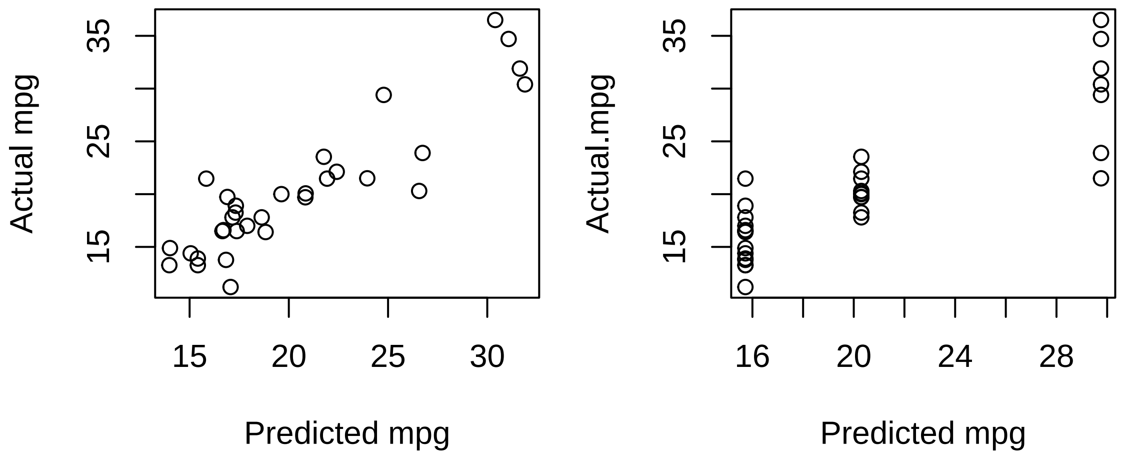


Figure 2.15: Random forest (left panel) and tree-based (right panel) predictions of car gas mileage.

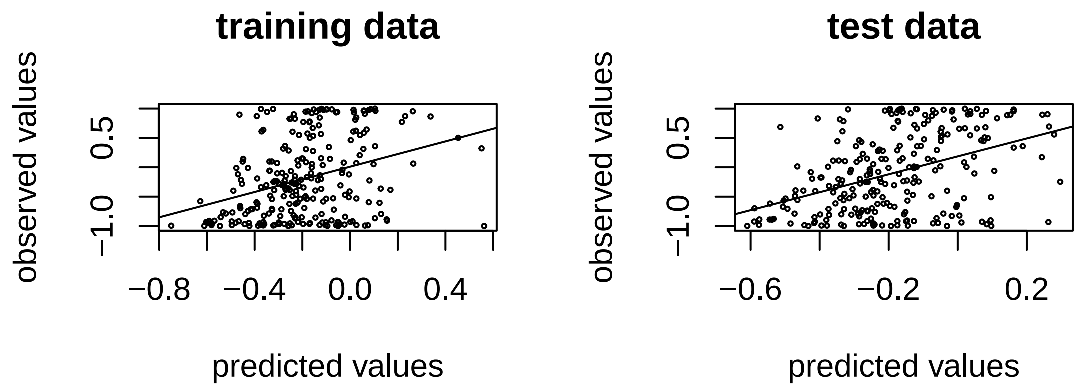


Figure 2.16: Random forest test for the cosine sequence of Section 2.1.2.

where m is the modulus; $m > 0$, a is the multiplier; $0 \leq a < m$, . c is the increment; $0 \leq c < m$, and x_0 is the starting value, or seed; $0 \leq x_0 < m$.

2.5.1 Conditions which prevent premature cycling

An interesting mathematical result concerning cycling in linear congruential pseudorandom number generators is given by Knuth (1997). It states that the linear congruential sequence defined by m, a, c and x_0 has cycle length m if and only if the following hold:

1. c is relatively prime to m (Two integers are relatively prime if there is no integer greater than one that divides them both (that is, their greatest common divisor is one). For example, 12 and 13 are relatively prime, but 12 and 14 are not.);

2. $b = a - 1$ is a multiple of p, for every prime p dividing m;

3. b is a multiple of 4, if m is a multiple of 4.

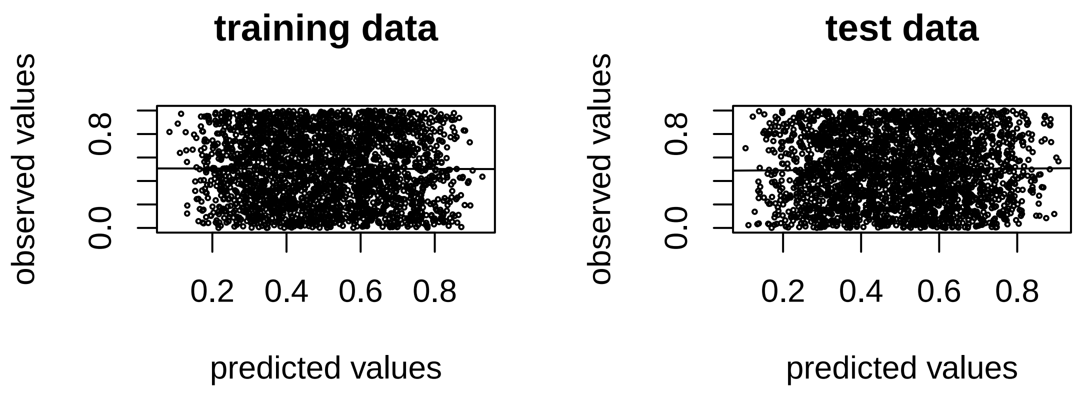


Figure 2.17: Random forest test for the default generator in R, using $m = 1$.

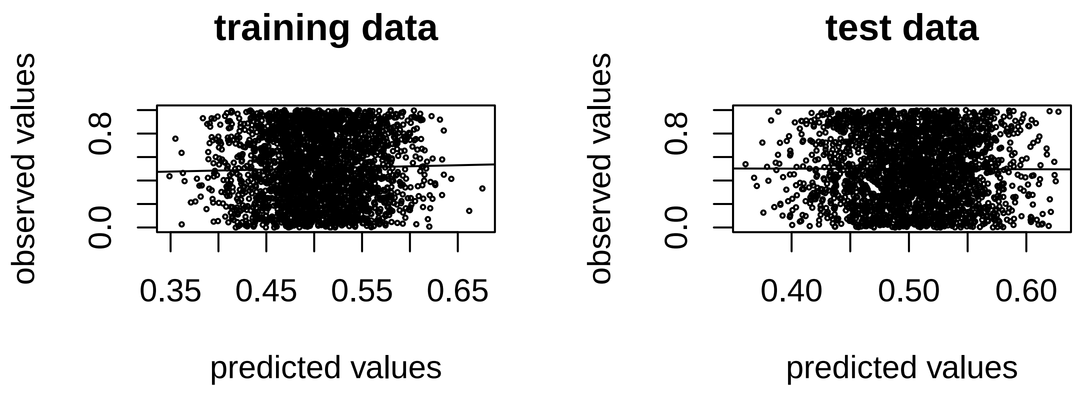


Figure 2.18: Random forest test for the default generator in R, using $m = 10$.

Example 2.16 *If $m = 8$, $b = 4$ and $x_0 = 3$, and $c = 3$. Then, $a = 5$ and the sequence is*

$$3, 2, 5, 4, 7, 6, 1, 0, 3$$

which has a cycle length 8.

2.5.2 Implementation

The following R function implements the linear congruential method:

```
rlincong <- function(n, m = 2^16, a = 2^8+1 , c = 3, seed) {
    x <- numeric(n)
    xnew <- seed
    for (j in 1:n) {
        xnew <- (a*xnew + c)%%m
        x[j] <- xnew
```

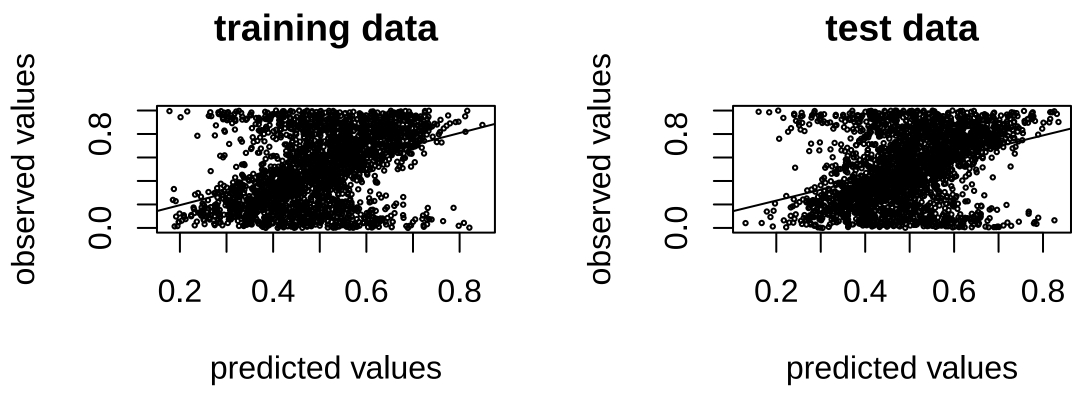


Figure 2.19: Random forest test for the RANDU sequence, using $m = 2$.

```
    }
    x/m
}
```

Default settings are chosen to satisfy conditions of the theorem: $c = 3$ and m are relatively prime since they do not share prime factors, $a - 1$ is a multiple of 2 which is the only prime which divides m, and b is a multiple of 4 (m is a multiple of 4).

Example 2.17 *The following verifies the cycle length at the default settings and with seed 372737:*

```
u <- rlincong(2^16-1, seed = 372737)
u[1:4]

## [1] 0.691467 0.707138 0.734528 0.773636

length(unique(u)) - (2^16 - 1)

## [1] 0
```

A full cycle was achieved, which is what the theorem predicts.

Testing a linear congruential generator

The same testing procedures apply to linear congruential generators as for multiplicative congruential generators.

Example 2.18 *Figure 2.20 displays a histogram and the autocorrelation function for the example of Section 2.5.2. The histogram indicates that uniformity is not an issue with this sequence, and at short lags, the ACF values are acceptably small.*

```
par(mfrow=c(1,2))
hist(u); acf(u)
```

These tests indicate that the resulting numbers are appropriately uniformly distributed and that there is no short range linear dependence, but they are not rigorous enough for us to say this is a good generator; for example, see what happens when we check out larger lag autocorrelations.

The following code allows for checking the autocorrelations at larger lags:

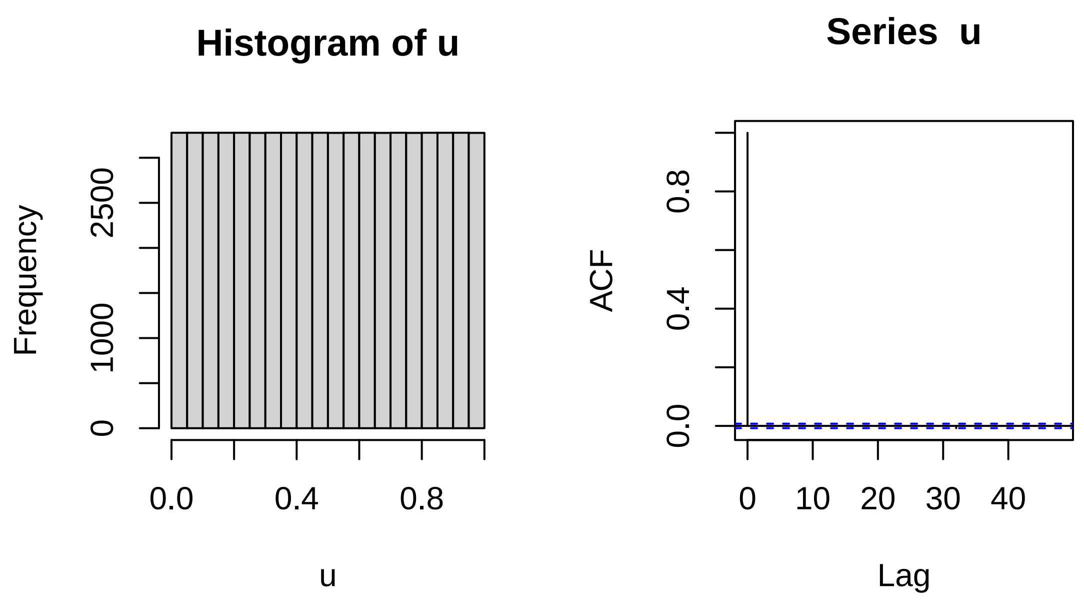


Figure 2.20: Histogram and ACF checks for the linear congruential sequence.

```
acf(u, lag.max =200)
```

Figure 2.21 shows that this generator is obviously far from perfect! Somehow, the value 129 lags earlier contains some information about the current value.

We can also apply the random forest test to the first 5000 elements of the sequence coming from the linear congruential generator:

```
u.sub <- u[1:5000]
rftest(u.sub,  m = 5)
```

The results displayed in Figure 2.22 confirm that this generator is not practical. The random forest makes very accurate predictions on the test data.

We conclude this discussion by noting that just because a pseudorandom number generator can have a long cycle length, it may yield numbers that are not sufficiently unpredictable. Basic histogram and autocorrelation function checks can quickly point out obvious problems, but for serious and extensive simulations involving very large sample sizes, it is necessary to employ more rigoroous tools, such as the spectral test. The random forest test appears to be a useful approximate tool.

2.6 Other pseudorandom number generators

2.6.1 A quadratic congruential generator

The following scheme has been proposed as an alternative to linear congruential generators. Let x_0 mod 4 = 2, and

$$x_{n+1} = x_n(x_n + 1) \bmod 2^{\omega}$$

where ω is a given positive integer.

Implementation as a function, with $\omega = 29$ by default, is as follows:

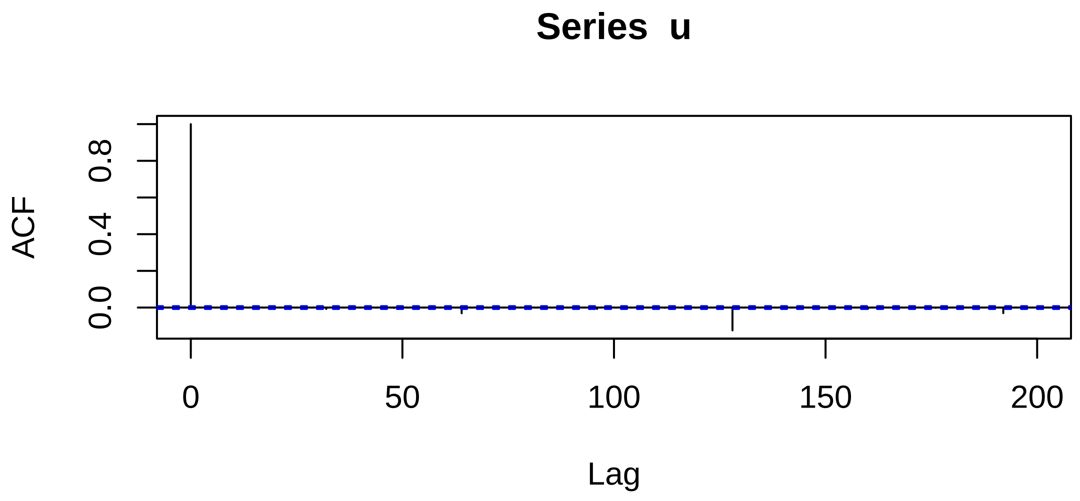


Figure 2.21: Histogram and ACF checks for the linear congruential sequence.

```
rquad <- function(n, seed, omega = 29) {
    if ((seed%%4)!=2) seed <- seed*4 + 2;  x0 <- seed
    numbers <- numeric(n)
    for (j in 1:n) {
        numbers[j] <- (x0*(x0+1))%%(2^omega)
        x0 <- numbers[j]
    }
    numbers/(2^omega)
}
```

Example 2.19 *We generate 1000 numbers using the default setting:*

```
quadnumbers <- rquad(1000, 39270)
```

The first few are

```
quadnumbers[1:8]

## [1] 0.872520 0.643710 0.209218 0.688178 0.654809 0.183330
## [7] 0.847959 0.722915
```

We can apply the histogram and autocorrelation function to look for obvious problems.

Example 2.20 *Figure 2.23 displays the histogram and ACF for the quadratic congruential sequence, using the default value of m.*

```
par(mfrow=c(1,2))
hist(quadnumbers); acf(quadnumbers)
```

The figure looks okay. Autocorrelations are near 0 and the histogram is flat.
If we generate another sequence, using $\omega = 10$, the results are much worse.

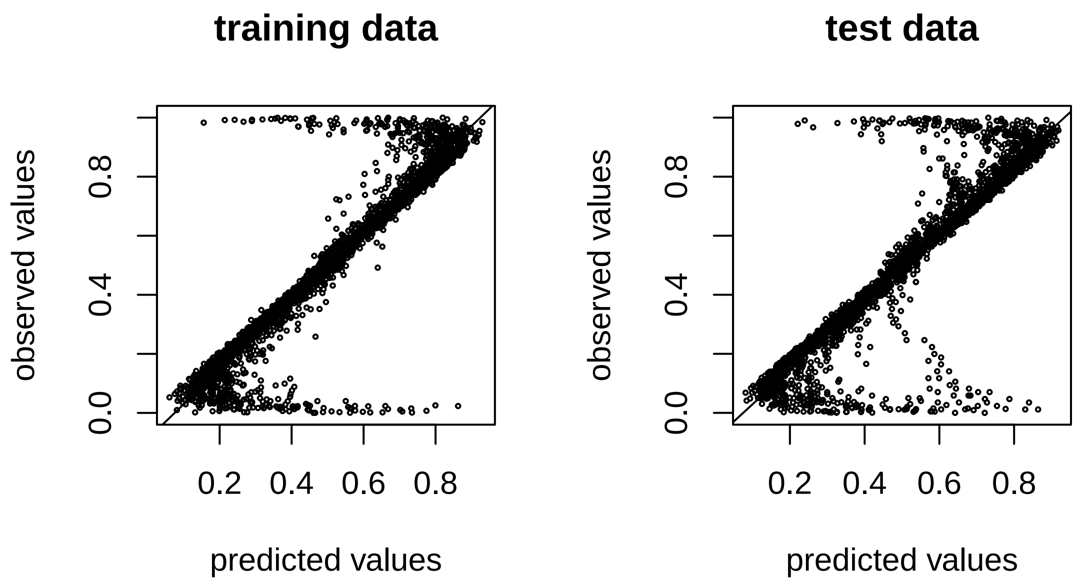


Figure 2.22: Random forest test for the linear congruential sequence, using $m = 5$.

```
par(mfrow=c(1,2))
prnumbers <- rquad(1000, 39270, 10) # different omega
hist(prnumbers); acf(prnumbers)
```

Figure 2.24 reveals some autocorrelations that are very large.

2.6.2 The Fibonacci method

Recall that the Fibonacci sequence is defined as

$$x_{n+1} = x_n + x_{n-1}$$

for $n = 1, 2, \ldots$, with $x_0 = 1$ and $x_1 = 1$.

The Fibonacci pseudorandom number generator is based on

$$x_{n+1} = (x_n + x_{n-1}) \bmod m$$

where x_0 and x_1 are taken as random seeds, and m is a given, large, positive integer.

Lüscher (1994) developed a class of generators which improves on this technique and are called luxury random number generators. These generators have been shown to possess superior statistical properties, passing the standard battery of tests.

2.6.3 The Mitchell and Moore method

The Fibonacci method is an example of an additive number generator. Another is based on the Mitchell and Moore sequence:

$$x_n = (x_{n-24} + x_{n-55}) \bmod m, n \geq 55$$

where m is a given, large, positive integer.

In this case, 55 integer seeds are needed in order to start the procedure.

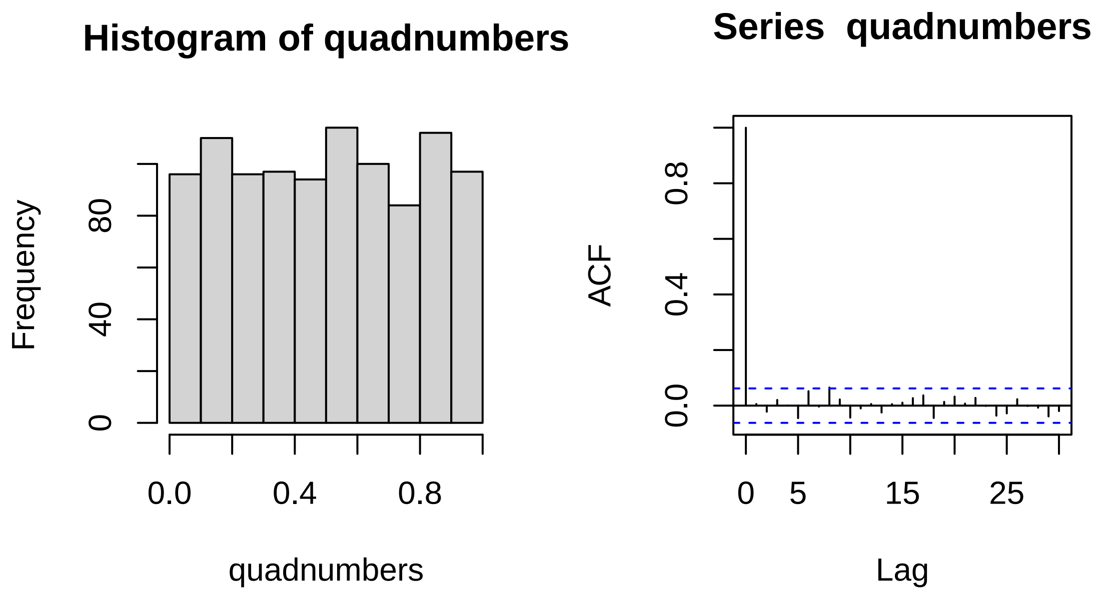


Figure 2.23: Histogram and ACF checks for the default quadratic congruential sequence.

2.6.4 Combined multiple recursive generator

The combined multiple recursive generator (cmrg) of L'Ecuyer (1996) is based on the difference of two underlying generators which are constructed from

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3}) \mod m_1$$

$$y_n = (b_1 y_{n-1} + b_2 y_{n-2} + b_3 y_{n-3}) \mod m_2$$

where $a_1 = 0$, $a_2 = 63308$, $a_3 = -183326$, $b_1 = 86098$, $b_2 = 0$, $b_3 = -539608$, $m_1 = 2^{31} - 1 = 2147483647$ and $m_2 = 2145483479$.

The simulated numbers are then given by

$$z_n = (x_n - y_n) \mod m_1.$$

2.6.5 An R generator: Wichman and Hill

One of the generators that is available for use in R is due to Wichman and Hill (1982). The numbers are obtained through taking the fractional part of the sum of pseudorandom numbers coming from three multiplicative congruential generators which have only moderate m values. It can be shown that the fractional part of the sum of three independent uniform random variables on $[0, 1]$ is, itself, a uniform random variable, supplying the theoretical justification for the method.

Checking for uniformity of sum of three uniforms mod 1

Figure 2.25 demonstrates that the fractional part of the sum of three uniform random variables is uniform. It should be clear that if successive values of the three sequences are independent, then the fractional parts are also sequentially independent. Figure 2.25 shows that the autocorrelations, even at very large lags, are all close to 0, a partial verification of this assertion.

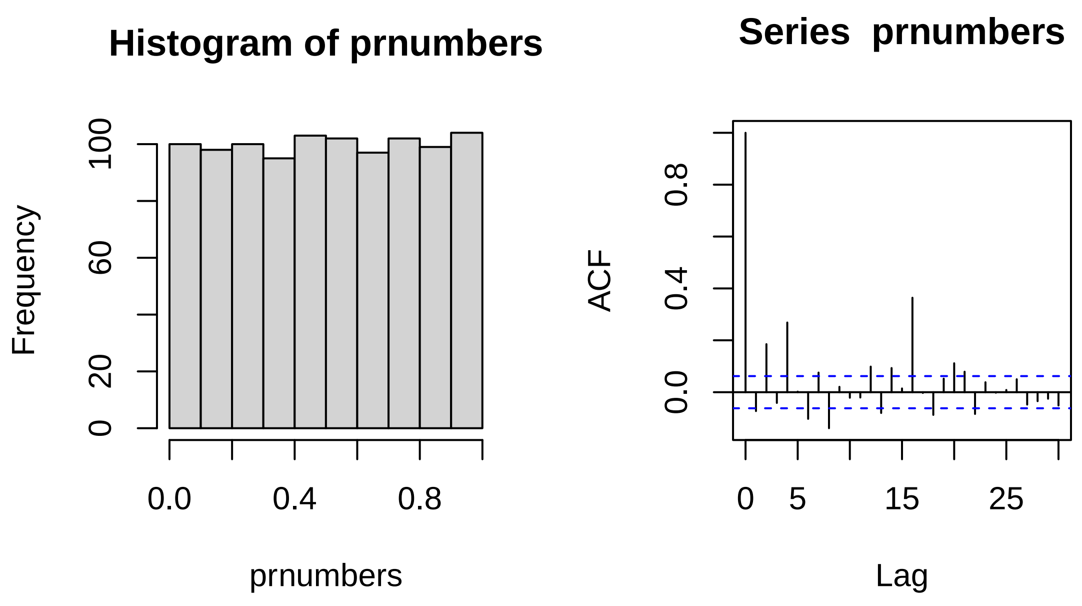


Figure 2.24: Histogram and ACF checks for another quadratic congruential sequence, with $\omega = 10$.

```r
u1 <- runif(10000)
u2 <- runif(10000)
u3 <- runif(10000)
par(mfrow=c(1,2))
hist((u1+u2+u3)%%1)
acf((u1+u2+u3)%%1, lag.max=1000)
```

Portability of Wichman and Hill's generator

The fact that it only requires the use of m values which are relatively small, provides a generator which is easy to implement on very modest computing systems.

Implementation of Wichman and Hill's generator

The following code shows how to implement the Wichman and Hill method in an R function.

```r
rWH <- function(n, seed) {
       if (missing(seed)) {
           seed <- 1000*as.numeric(paste(strsplit(
                format(Sys.time(), "%H:%M:%OS3"), ":")[[1]],
                collapse=""))
       }
       ix <- (171*seed)%%30269
       iy <- (172*seed)%%30307
       iz <- (170*seed)%%30323
       numbers <- numeric(n)
       for (i in 1:n) {
           ix <- (171*ix)%%30269
```

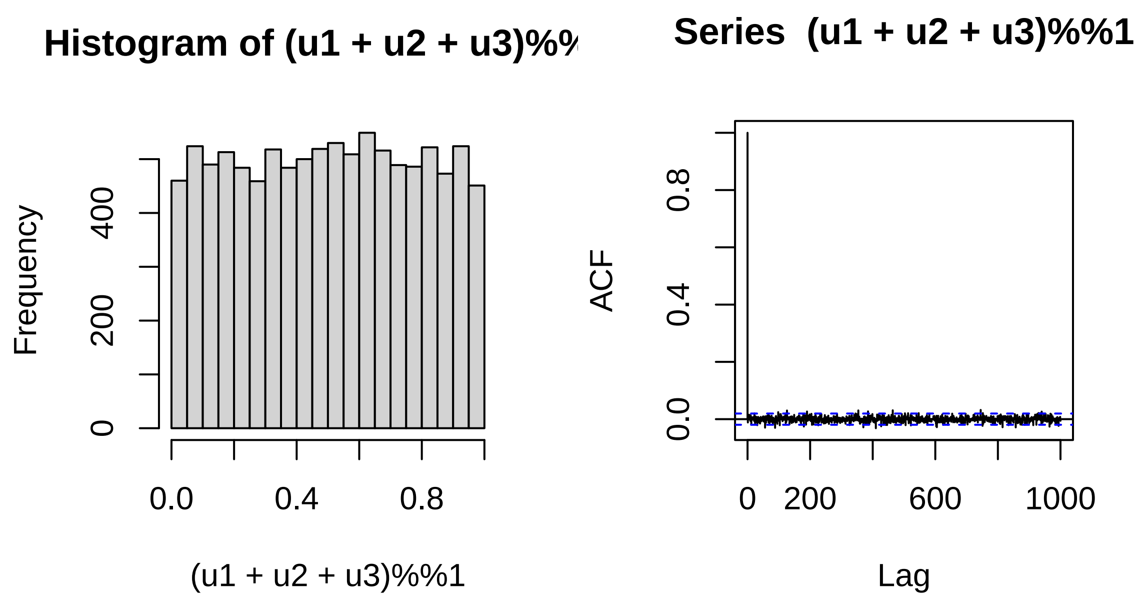


Figure 2.25: The fractional part of sums of independent uniforms is uniform.

```
            iy <- (172*iy)%%30307
            iz <- (170*iz)%%30323
            numbers[i] <- (ix/30269.0+ iy/30307.0+
                  iz/30323.0)%%1.0
        }
        numbers
}
```

Example 2.21 *We illustrate the use of the function to produce Wichman and Hill sequences here:*

```
rWH(4) # automatically generated seed

## [1] 0.160200 0.892577 0.729546 0.128418

rWH(4)

## [1] 0.05545427 0.00405712 0.66718409 0.93365718

rWH(4, 3329913) # our own seed

## [1] 0.637332 0.400932 0.493218 0.261488
```

Checking for autocorrelation and uniformity proceeds as before.

Example 2.22
```
par(mfrow=c(1,2))
prnumbers <- rWH(10000)
hist(prnumbers); acf(prnumbers)
```

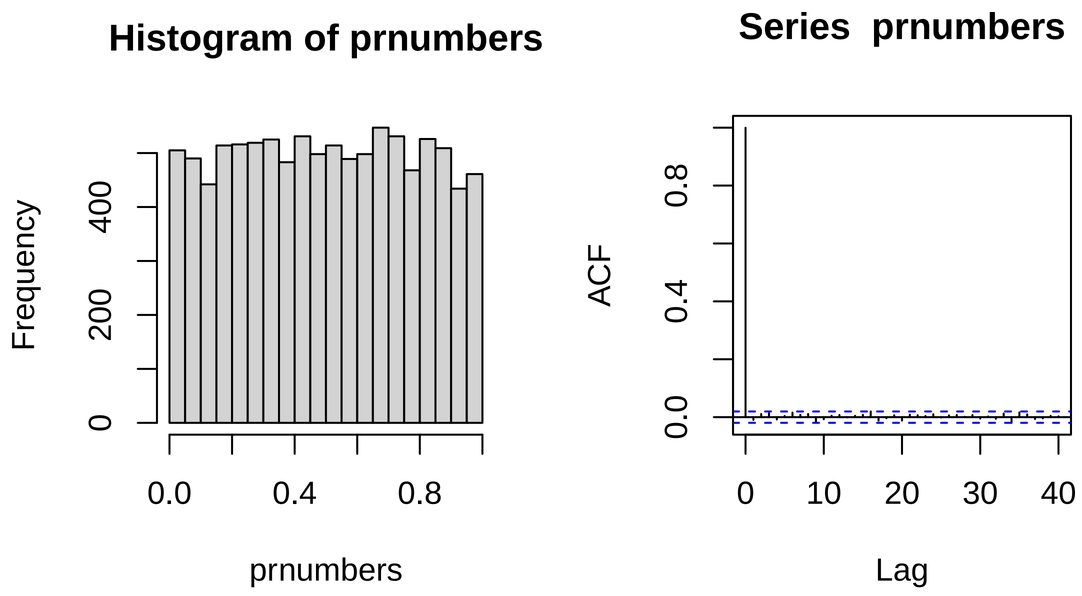


Figure 2.26: Histogram and ACF checks for the Wichman and Hill generator.

Figure 2.26 contains the histogram and autocorrelation function for a sample of 10000 Wichman and Hill numbers. The ACF is small, and the histogram is flat

Further checks would, of course, be necessary to confirm the usefulness of this generator.

2.6.6 Theory

```
##  [1] "H" "H" "T" "H" "T" "H" "H" "T" "T" "T" "T" "T" "T" "H"
## [15] "T" "H" "T" "T" "H" "H" "H" "H"
```

Table 2.8 displays the frequency of occurrence for each possible outcome. The table shows that if one chooses the seed randomly from the set $\{1, 2, \ldots, 16\}$, a pair of heads will occur with probability 5/16 as is the case for a pair of tails. On the other hand, H-T and T-H pairs occur with probability 3/16 each. Thus, the generator will give a biased result for this simple problem, since we should be expecting to see any of these four outcomes with exactly the same probability.

Restricting away from seeds 1 and 22 will result in perfect simulation of two independent coin tosses.

Issues with choice of seeds - in generators such as Wichman and Hill where multiple seeds are required for ”independent” generators and for distributed parallel computation - reference (Savage et al., 1994; Davies and Brooks, 2007; Warne et al., 2021; **?**)

Include induction proof here.

2.7 Default generator in R: the Mersenne twister

Matsumoto and Nishimura (1998) are responsible for the default pseudorandom number generator used in R. It is named for the fact that its cycle length is exactly a Mersenne prime (a prime number of the form $2^p - 1$, for example, $2^3 - 1 = 7$). The cycle length in the 32-bit implementation is $2^{19937} - 1$.

Its theoretical basis is more complicated than congruential approaches, and it passes most statistical tests for randomness. However, it is slow by modern standards.

seed	outcome
1	H H
7	H T
3	T H
21	H T
9	T H
17	H H
4	H T
5	T T
12	T T
15	T T
12	T T
22	T T
16	T H
20	H T
2	T H
14	H T
6	T T
19	T H
18	H H
11	H H
8	H H
10	H H

Table 2.7: Simulated pairs of coin tosses for given seeds.

H H	6
H T	5
T H	5
T T	6

Table 2.8: Observed frequencies of successive coin toss outcomes

To invoke this generator in R to produce a sequence of n uniform numbers in the interval $[a, b]$, use

```
runif(n, min = a, max = b)
```

The default values are $a = 0$ and $b = 1$. The seed is selected internally.

Example 2.23 *Generate 3 uniform pseudorandom numbers on the interval* $[0, 1]$ *and 5 uniform such numbers on the interval* $[-2, 3]$.

```
runif(3)

## [1] 0.442134 0.504272 0.334078
```

```
runif(10, min = -2, max = 3)

##  [1]  2.334854  1.352881  0.618596  0.177228 -1.448725
##  [6] -1.283657 -0.319182  1.390607  2.495899  0.211675
```

2.8 Initial seeds

If you run the code provided in this chapter to generate pseudorandom numbers yourself, you will most likely obtain different results than those displayed in the text. This is because your initial seed is likely different from the one used in the production of this text.

There are two ways of selecting a starting seed x_0. If your objective is to produce an entirely unpredictable sequence, then you should try to start with some kind of unpredictable value, such as the current time, measured in very precise units. You should be careful not to re-seed, since you could inadvertently introduce predictability, say, by seeding at exactly the same time every day.

The second strategy for choosing a seed is to use a particular value, e.g. $x_0 = 26636$. If you use such a value, and keep a record of it, then you can reproduce your entire simulation by invoking this seed before re-executing your code. All output should be the same, if the exact same sequence of statements are executed each time.

In R, use the `set.seed()` function to fix the initial seed for the pseudorandom number generators used there.

Example 2.24 *If you run the following code without specifying the seed, you will likely obtain values different from the 6 obtained below:*

```
runif(6)

## [1] 0.194741 0.671348 0.362032 0.140283 0.959180 0.449769
```

If you run the following two lines of code, you will obtain exactly the same set of 6 numbers:

```
set.seed(336600)
runif(6)

## [1] 0.651550 0.733344 0.835543 0.290004 0.358213 0.962804
```

2.9 Shuffling

Analogous to shuffling a deck of cards, there is a shuffling technique for reducing sequential dependence in a given sequence of pseudorandom numbers, x.

The shuffling algorithm uses an auxiliary table $v(1), v(2), \ldots, v(k)$, where k is some number chosen arbitrarily, usually in the neighborhood of 100. Initially, the v vector is filled with the first k values of the x sequence and an auxiliary variable y is set equal to the $(k+1)st$ value. The steps are:

Extract the index j . Set $j \leftarrow ky/m$, where m is the modulus used in the sequence x; that is, j is a random value, $0 \leq j < k$, determined by y.

Exchange. Set $y \leftarrow v[j]$, return y, and set $v[j]$ to the next member of the sequence x.

The following is an R implementation of shuffling, using the built-in generator to generate the auxiliary sequence.

```
shuffle <- function(n, k = 100, x = runif(n)) {
    v <- x[1:k]
    y <- x[k+1]
    xnew <- numeric(n - k)
    i <- 1
    while (n > k) {
        j <- floor(k*y)+1
        y <- v[j]
        xnew[i] <- y
        i <- i + 1
        v[j] <- x[k+1]
        x[k+1] <- x[n]
        n <- n-1
    }
    c(v, xnew)
}
```

Example 2.25 ***Shuffling a deck of 52 playing cards.***
We first define a vector containing the 52 different playing cards, using a factor called `cards` *with 52 levels:*

```
a <- 1:52
suits <- c("Spades", "Hearts", "Diamonds", "Clubs")
values <- c(2:10, "J", "Q", "K", "A")
cards <- factor(a)
levels(cards) <- as.vector(outer(values, suits, paste))
cards # sorted

##  [1] 2 Spades    3 Spades    4 Spades    5 Spades
##  [5] 6 Spades    7 Spades    8 Spades    9 Spades
##  [9] 10 Spades   J Spades    Q Spades    K Spades
## [13] A Spades    2 Hearts    3 Hearts    4 Hearts
## [17] 5 Hearts    6 Hearts    7 Hearts    8 Hearts
## [21] 9 Hearts    10 Hearts   J Hearts    Q Hearts
## [25] K Hearts    A Hearts    2 Diamonds  3 Diamonds
## [29] 4 Diamonds  5 Diamonds  6 Diamonds  7 Diamonds
## [33] 8 Diamonds  9 Diamonds  10 Diamonds J Diamonds
## [37] Q Diamonds  K Diamonds  A Diamonds  2 Clubs
## [41] 3 Clubs     4 Clubs     5 Clubs     6 Clubs
## [45] 7 Clubs     8 Clubs     9 Clubs     10 Clubs
## [49] J Clubs     Q Clubs     K Clubs     A Clubs
## 52 Levels: 2 Spades 3 Spades 4 Spades 5 Spades ... A Clubs
```

We can shuffle the cards using our `shuffle()` *function:*

```
a <- as.numeric(cards)/53
par(mfrow=c(3,3))
for (i in 1:9) {
    ts.plot(a)
    a <- shuffle(52, 10, a)
}
```

```
cards[a*53] # shuffled

##  [1] J Hearts    10 Spades   K Spades    K Diamonds
##  [5] 4 Hearts    6 Spades    3 Clubs     2 Hearts
##  [9] 5 Spades    7 Spades    8 Clubs     8 Hearts
## [13] 10 Diamonds 6 Clubs     A Spades    Q Clubs
## [17] 9 Diamonds  9 Spades    J Spades    A Clubs
## [21] J Diamonds  A Diamonds  9 Hearts    5 Hearts
## [25] 6 Hearts    7 Hearts    3 Diamonds  2 Diamonds
## [29] 10 Hearts   7 Clubs     9 Clubs     Q Spades
## [33] 7 Diamonds   4 Clubs      4 Diamonds  5 Diamonds
## [37] Q Diamonds   3 Spades     J Clubs     K Clubs
## [41] 6 Diamonds   4 Spades     8 Diamonds  2 Clubs
## [45] Q Hearts     3 Hearts     5 Clubs     8 Spades
## [49] 10 Clubs     A Hearts     2 Spades    K Hearts
## 52 Levels: 2 Spades 3 Spades 4 Spades 5 Spades ... A Clubs
```

Figure 2.27 shows how the original ordering of the numbers from 1 through 52 gradually takes on more of an unpredictable appearance as the number of shuffles increases.

Next, we can ask how many times we should shuffle to obtain a reasonably random ordering of the cards:

```
a <- as.numeric(cards)/53
par(mfrow=c(3,3))
for (i in 1:9) {
    acf(a)
    a <- shuffle(52, 10, a)
}
```

Figure 2.28 displays the autocorrelations for the shuffled numbers. The autocorrelations appear to die down after the numbers have been shuffled 6 or 7 times.

The cross-correlation between an n-vector x and another n-vector y, at lag m essentially measures the correlation between $(x_1, x_2, \ldots, x_{n-m})$ and $(y_m, y_{m+1}, \ldots, y_n)$.

Example 2.26 *We can use the cross-correlation function to see how much dependence occurs between "hands". Figure 2.29 contains graphs of the cross-correlations between the original ordering of the 52 numbers and the orderings following each of 9 successive shuffles of those numbers.*

```
b <- a # b contains the order for the original hand
# a will contain the order for the next 9 shuffles:
par(mfrow=c(3,3))
for (i in 1:9) {
    a <- shuffle(52, 10, a)
    ccf(a, b)
}
```

Comment about weaknesses - see Anderson

2.10 Fast computation

check this section

Refer to Anderson (1990) for vectorized algorithm.

Use of the `modpower()` function in the *numbers* package (Borchers, 2021).

```
library(numbers)
rngV <- function(n, a, cc, m, seed, L) {
    x <- numeric(n)
    ai <- numeric(L+1)
    ell <- 0:L
    for (i in ell) {
        ai[i+1] <- modpower(a, ell[i+1], m)
}
```

```
    ci <- (cc*cumsum(ai[-(L+1)]))%%m
    ai <- ai[-1]
    x[1:L] <- (ai*seed  + ci)%%m
    M <- ceiling(n/L)
    for (j in 1:(M-1)) {
        seed <- x[j*L]
        x[(j*L+1):((j+1)*L)] <- (ai*seed + ci)%%m
    }
    return(x[1:n]/m)
}
```

The multiplicative congruential generator with modulus $m = 2^{19} - 1$ and $a = 701$ has a cycle length of $2^{19} - 2$ as shown below.

```
length(unique(rngV(524286, 701, 0, 524287, 1, 700)))

## [1] 524286
```

We can check that the vectorized generator produces the exact same sequence as the sequential version by computing the pairwise differences of the results and computing the mean of the absolute difference.

```
mean(abs(rngV(524286, 701, 0, 524287, 1, 700) - rng(524286, 701, 524287)))

## [1] 0
```

The time taken to generate the entire cycle with the vectorized version is about 25% of the sequential version.

```
system.time(rngV(524286, 701, 0, 524287, 1, 700))

##    user  system elapsed
##   0.024   0.000   0.024
```

```
system.time(rng(524286, 701,  524287))

##    user  system elapsed
##   0.111   0.000   0.110
```

2.10.1 Recently developed generators

A number of new generators have been developed since the advent of the Mersenne Twister. Some of these are discussed by Gevorkyan et al. (2020). More detail on random number testing is given there, including information on the Diehard and Dieharder set of statistical tests for random number generators. They also refer to three modern generators that are simpler and hence faster than the Mersenne Twister. They comment that the Mersenne Twister is awkward but generally has good properties.

The XORshift generators of Panneton and L'ecuyer (2005) are fast but seem to fail statistical tests. The permuted congruential generators (PCG) were introduced by O'Neill (2014). These generators use a randomized algorithm which adds a level of security which is not a feature of most other generator families.

The PCG family of generators has been ported into R using the Rcpp function through the *dqrng* package (Stubner, 2021). The package also contains ports to the `Xoshiro256+` and `Xoroshiro128+` pseudorandom number generators which are due to Blackman and Vigna (2021). The latter is the fastest generator available in the *dqrng* package.

```
library(dqrng)
```

```
dqset.seed(42) # set the seed
```

```
dqRNGkind("Xoshiro256+")
uniforms <- dqrunif(5000)
hist(uniforms)
```

```
library(microbenchmark)
microbenchmark(dqrunif(1000000))

## Unit: milliseconds
##            expr     min     lq   mean median      uq
##  dqrunif(1e+06) 3.63794 3.6563 3.9511 3.6941 3.73811
##      max neval
##  8.65264   100
```

```
dqRNGkind("Xoroshiro128+")
microbenchmark(dqrunif(1000000))

## Unit: milliseconds
##            expr     min      lq   mean  median      uq
##  dqrunif(1e+06) 3.37459 3.38902 3.7547 3.41964 3.47753
##      max neval
##  8.39973   100
```

The `pcg64` generator:

```
dqRNGkind("pcg64")
microbenchmark(dqrunif(1000000))

## Unit: milliseconds
##            expr     min     lq    mean median      uq
##  dqrunif(1e+06) 4.13967 4.1436 4.50017 4.1649 4.22785
##      max neval
##  9.40602   100
```

```
microbenchmark(runif(1000000))

## Unit: milliseconds
##          expr     min      lq    mean  median      uq
##  runif(1e+06) 23.0309 23.1303 23.6476 23.2371 23.5988
##      max neval
##  28.0841   100
```

```
dqRNGkind("pcg64")
```

2.10.2 Use of the Gnu Scientific Library (GSL) generators

Wickham (2014) makes a compelling case for the use of the *Rcpp* facility in R to interface with C++ and the GSL library (The GSL Team, 2021) to speed up code, particularly random number generation. In order to install *RcppGSL* (Eddelbuettel and Francois, 2022), you need to have a working version of GSL. On a computer running a Linux (Debian) operating system, this can be installed using

```
sudo apt install libgsl-dev
```

The package *gsl* (Hankin, 2006) provides a facility for accessing these generators without needing to program in C++. Setting up the cmrg generator (see Section 2.6.4) is as follows:

```
library(gsl)
r <- rng_alloc("cmrg")
rng_set(r, 100)

## [1] 100

rcmrg <- function(n) rng_uniform(r, n)
```

We can then use the function `rcmrg()` in the same way that we would use `runif()`. For example, generating 10 numbers proceeds as

```
rcmrg(10)

##  [1] 0.75100266 0.27632556 0.80290789 0.79234885 0.00991752
##  [6] 0.90312322 0.14127554 0.44023898 0.50391344 0.88495743
```

We now compare the speed performance of cmrg with R's implementation of the Mersenne Twister.

```
microbenchmark(x <- rcmrg(1000000))

## Unit: milliseconds
##                 expr     min       lq     mean median       uq
##  x <- rcmrg(1e+06) 18.9881 19.0127 19.5238 19.051 19.3047
##      max neval
##  21.6693   100

microbenchmark(x <- runif(1000000))

## Unit: milliseconds
##                 expr     min       lq     mean  median       uq
##  x <- runif(1e+06) 22.9973 23.0985 23.5612 23.1415 23.4043
##      max neval
##  25.5599   100
```

The luxury random number generators or `ranlux` algorithms (James, 1994) are also available in GSL. One of the faster ones is `ranlxs0`.

```
r <- rng_alloc("ranlxs0")
rng_set(r, 100)

## [1] 100

rlxs0 <- function(n) rng_uniform(r, n)
```

```
microbenchmark(x <- rlxs0(1000000))

## Unit: milliseconds
##                expr     min      lq    mean  median      uq
##   x <- rlxs0(1e+06) 22.7481 22.7977 23.4151 22.9311 23.8434
##       max neval
##   25.4986   100
```

2.11 Further reading

Knuth (1997) offers a comprehensive treatment of pseudorandom number generation and testing which remains authoritative. Law et al. (2007) provides a somewhat more up-to-date treatment, noting some more recent methods, for example. Another older, but very important reference, is due to Anderson (1990).

Exercises

1. Consider the following iteration scheme:

$$x_{n+1} = f(x_n) := \frac{2x_n}{3} + \frac{16}{x_n^2}$$

where x_0 is the initial value, say, $x_0 = 28$.

   (a) Plot the graph of the function $f(x)$, for $x \in [1, 5]$ and overlay the straight line with intercept 0 and slope 1. Does $f(x)$ have a fixed point? Solve the equation $x = f(x)$ for x to determine its value.

   (b) Using a `for` loop in R, run 10 steps of the proposed scheme, printing out the value of x_n at each step.

   (c) Based on your analysis, could the proposed scheme lead to a useful pseudorandom number generator?

2. Consider the following iteration scheme:

$$x_{n+1} = f(x_n) := x_n + 7\mathrm{e}^{-x_n} - 1$$

where x_0 is the initial value, say, $x_0 = 2$.

   (a) Plot the graph of the function $f(x)$, for $x \in [0, 5]$ and overlay the straight line with intercept 0 and slope 1. Does $f(x)$ have a fixed point? Solve the equation $x = f(x)$ for x to determine its value.

   (b) Using a `for` loop in R, run 10 steps of the proposed scheme, printing out the value of x_n at each step.

   (c) Based on your analysis, could the proposed scheme lead to a useful pseudorandom number generator?

3. Consider the following iteration scheme:

$$x_{n+1} = f(x_n) := \frac{x_n}{2} - \frac{24.5}{x_n}$$

where x_0 is the initial value, say, $x_0 = 0.5$.

   (a) Plot the graph of the function $f(x)$, for $x \in [0.1, 10]$ and overlay the straight line with intercept 0 and slope 1. Does $f(x)$ have a fixed point?

   (b) Using a `for` loop in R, run 10 steps of the proposed scheme, printing out the value of x_n at each step.

   (c) Based on your analysis, could the proposed scheme lead to a useful pseudorandom number generator?

4. Consider the following iteration scheme:

$$x_{n+1} = f(x_n) := \frac{x_n}{2} - \frac{24.5}{\sqrt{x_n}} \mod 1$$

where x_0 is the initial value, say, $x_0 = 0.5$.

   (a) Plot the graph of the function $f(x)$, for $x \in [0.01, 1]$ and overlay the straight line with intercept 0 and slope 1. Does $f(x)$ have a fixed point?

   (b) Using a `for` loop in R, run 10 steps of the proposed scheme, printing out the value of x_n at each step.

   (c) Based on your analysis, could the proposed scheme lead to a useful pseudorandom number generator?

5. Construct a function which takes `n` as an argument and implements the pseudorandom number generator iteration for the preceding question, returning, as output, a vector of length `n`. Conduct the basic checks on the quality of this generator? If it passes those basic checks, apply the random forest test to determine whether this generator could be used, at least in small simulations, using up to 2000 numbers.

6. Which of the following linear congruential pseudorandom number generators have maximal cycle length?

   (a) $a = 1025, c = 27, m = 2^{31}$

   (b) $a = 1025, c = 54, m = 2^{31}$

   (c) $a = 1025, c = 375, m = 2^{31}$

   (d) $a = 10241, c = 375, m = 2^{31}$

7. Run basic checks on each of the generators from the previous question to determine whether any are obviously defective. For any that aren't, apply the random forest test for $m \le 10$ as a further check. Do you think any of these generators could be used, at least for small problems?

8. Write an R function that implements the Fibonacci pseudorandom number generator, taking `n` and `x0` as the arguments and returning a vector of `n` pseudorandom numbers. Run the basic tests as well as the random forest test to assess the quality of this generator.

9. Write an R function that implements the Mitchell and Moore pseudorandom number generator, taking `n` and `x0` as the arguments and returning a vector of `n` pseudorandom numbers. Note that the seed is necessarily a vector of length 55. Run the basic tests as well as the random forest test to assess the quality of this generator.

10. Consider the following random number generator which is based purely on shuffling an initially ordered sequence.

```
a <- (1:1000)/1001
for (i in 1:20) {
   a <- shuffle(1000, 50, a)
}
a
```

   (a) Test the sequence for uniformity. Write out your conclusion clearly.

   (b) Check the autocorrelations. Is there any indication of sequential linear dependence?

   (c) Construct a lag plot as a way of applying the spectral test in two dimensions. What do you conclude? What would you predict about the 51st number if you knew the 50th number was 0.5?

11. Repeat the previous question, but start with `a` being the first 1000 numbers generated from a multiplicative congruential generator with $b = 171$ and $m = 30269$. Also, check the autocorrelations for the multiplicative generator and compare with the performance after shuffling; does shuffling help?

12. Write an R function which applies the shuffling algorithm to output from the RANDU generator. Run the random forest test on sequences coming from the new generator to determine if you have an improved generator.

13. Construct an R function called `ranlux389()` that implements the GSL `ranlux389` generator using the tools provided in the *gsl* package. Run a time comparison between your new generator and `runif()` to see which is faster.

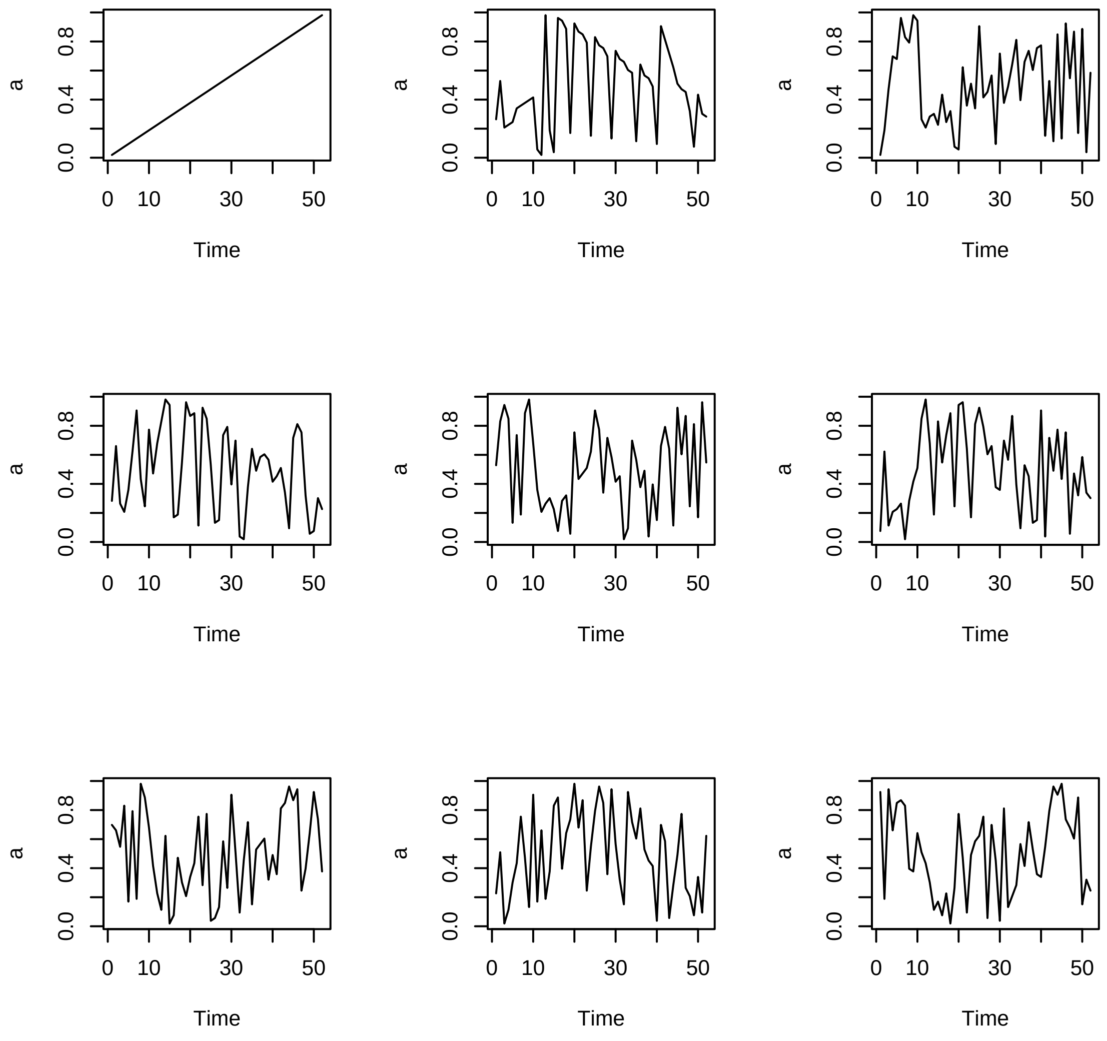


Figure 2.27: Trace plots for 9 successive shuffles of a deck of 52 numbers (representing playing cards in an ordinary deck).

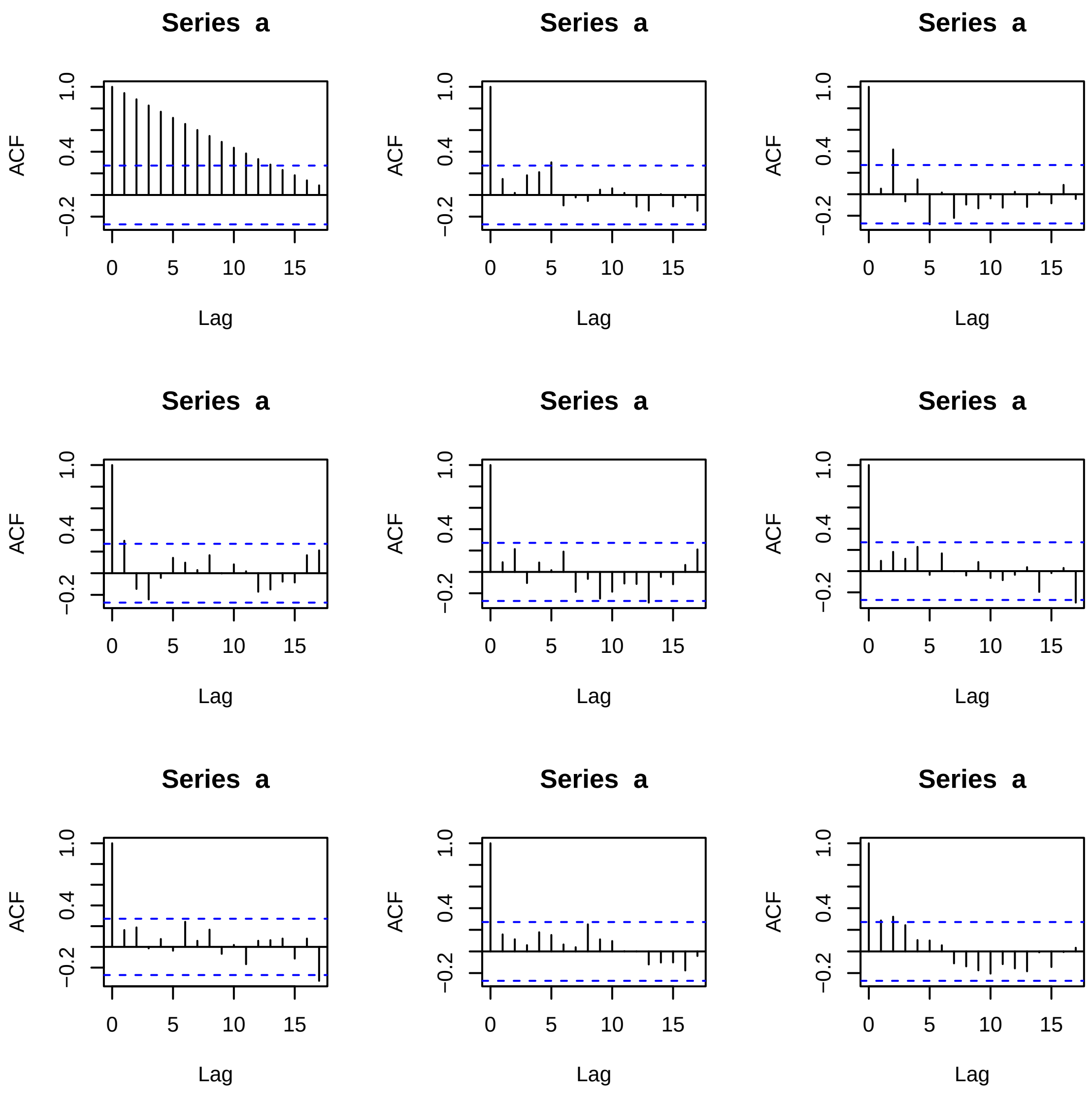


Figure 2.28: Autocorrelations for 9 successive shuffles of a deck of 52 numbers (representing playing cards in an ordinary deck).

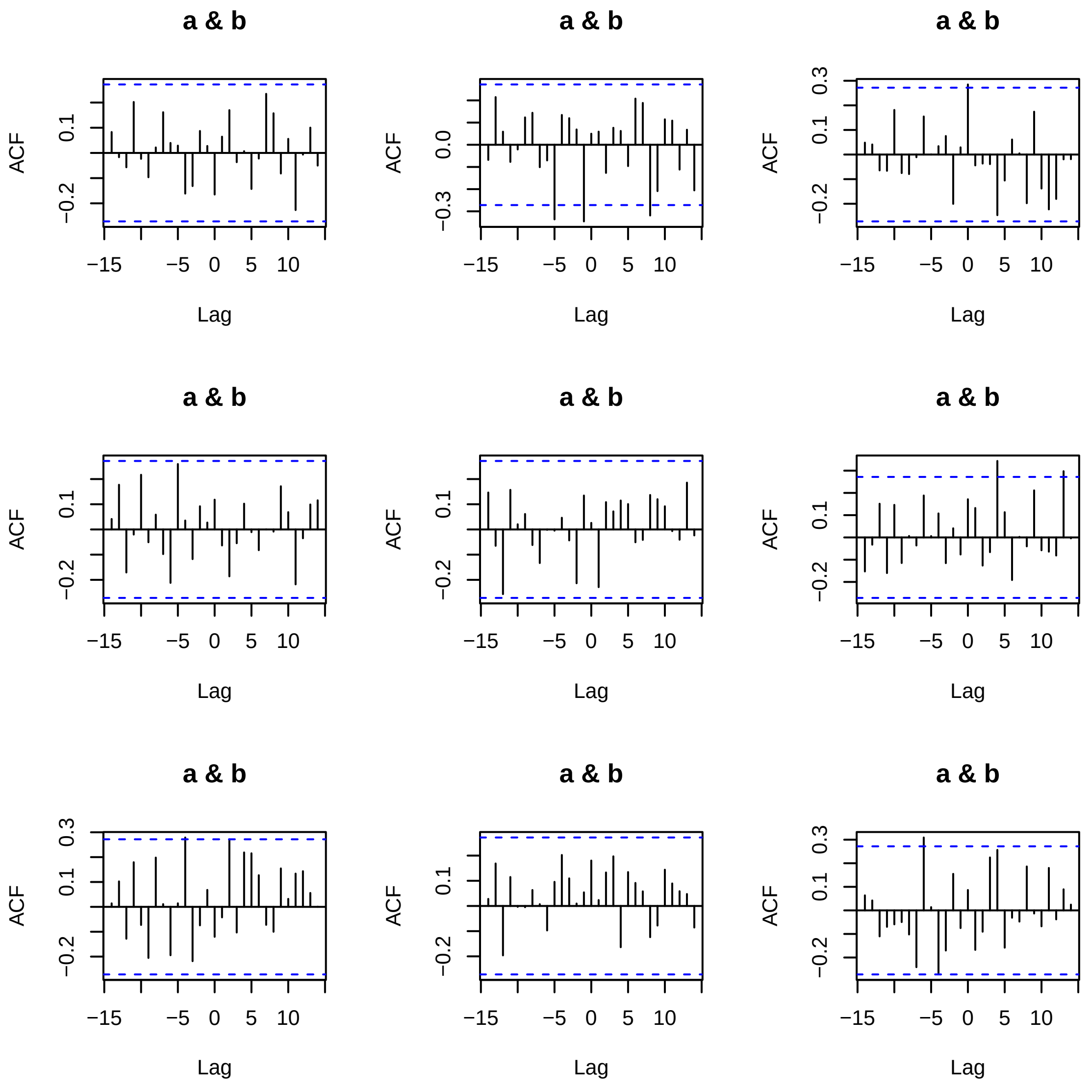


Figure 2.29: Cross-correlations between original ordering and first 9 orderings of a shuffled deck of 52 numbers (representing playing cards in an ordinary deck).

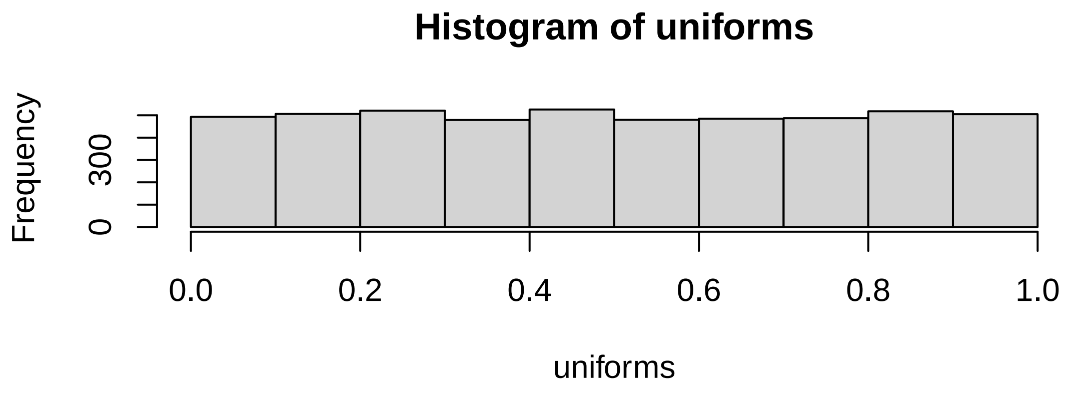


Figure 2.30: Histogram of 5000 extttXoshiro256+ pseudorandom numbers.

3

Simulation of Discrete Random Variables

This chapter is concerned with basic Monte Carlo simulation methods for generating pseudorandom numbers which follow different kinds of probability distributions.

3.1 Discrete random variables

Discrete random variables are characterized in the following way. They take on numeric values. The set of possible values for a discrete random variable is either finite or countably valued. They are often, but not always, integer-valued. Each value is associated with a probability.

The following are examples for which discrete random variables can provide useful models:

1. The number of major earthquakes in a region.
2. The number of errors in a software program.
3. The number of accidents at a traffic intersection.
4. The proportion of mosquitoes killed at a given dose of insecticide.
5. The number of attempts made at passing a test until the first success.

Probabilities for discrete random variables can sometimes be inferred logically. Applications are usually quite simplistic.

A function $p_X(x)$ specifies the probability that the random variable, X, takes on the value x. We write $p_X(x) = P(X = x)$.

Example 3.1 *Consider the number of heads H obtained in 2 independent coin tosses. The possible values of H are in the set $\{0, 1, 2\}$. The corresponding probabilities are $p_H(0) = 1/4, p_H(1) = 1/2$, and $p_H(2) = 1/4$.*

The values of a discrete random variable, together with the associated probabilities is referred to as a probability distribution. Such a distribution can also be summarized in a table.

Example 3.2 *The table of probabilities for outcomes of two coin tosses is as follows:*

h	0	1	2
$p_H(h)$	1/4	1/2	1/4

Probability distributions can also be modelled from data. This is a very common occurrence.

Example 3.3 *For quality purposes, in a refrigerator manufacturing setting, the number of flaws were counted in the surfaces of a sample of 100 refrigerator doors, yielding the following information:*

Number of Flaws	0	1	2	3	4	5	6
Count	35	39	12	13	0	0	1

The sample can be viewed as an estimate for the total population. If we divide the sample counts by the total sample size, we obtain proportions for the various values: estimates of the theoretical probabilities of the various counts.

Example 3.4 *We can divide by the sample size, 100, to yield an estimate of the probability distribution for surface flaws F:*

f	0	1	2	3	4	5	6
$d_F(f)$	0.35	0.39	0.12	0.13	0	0	0.01

In other words, there is an approximate probability of 0.12 that a randomly selected refrigerator door has exactly 2 surface flaws. Note the use of the notation $d_F(f)$ to denote the function which associates the probability values with the various outcome values.

We could create a function in R to return such probabilities:

```r
dFlaws <- function(x) {
    return(c(.35, .39, .12, .13, 0, 0, .01)[x+1])
}
```

e.g. $P(F = 3)$:

```r
dFlaws(3)

## [1] 0.13
```

3.1.1 Visualizing a discrete distribution

The bar plot is an appropriate vehicle to view discrete distributions.

Example 3.5 *Displaying the observed surface flaw distribution proceeds as follows:*

```r
f <- 0:6
probs <- dFlaws(f)
names(probs) <- f
barplot(probs)
```

Figure 3.1 displays the resulting bar plot.

In the code for this example, we have converted the numeric vector `probs` into a named vector, where each element is associated with a name, which is simply a character which has been coerced from the numeric vector `f`. The `barplot()` function uses those names for the horizontal axis ticklabels.

3.1.2 Cumulative Distributions

For a given random variable X, we often need to evaluate $P(X \leq x)$ or its complement $P(X > x) = 1 - P(X \leq x)$.

The cumulative distribution function is $F(x) = P(X \leq x) = \sum_j^x P(X = j)$.

Example 3.6 *For the surface flaws example, we would estimate the cumulative distribution function as:*

```r
pFlaws <- function(x)  {
    return(c(.35, .74,.86, .99, .99, .99, 1)[x+1])
}
```

For example, $P(F > 4)$

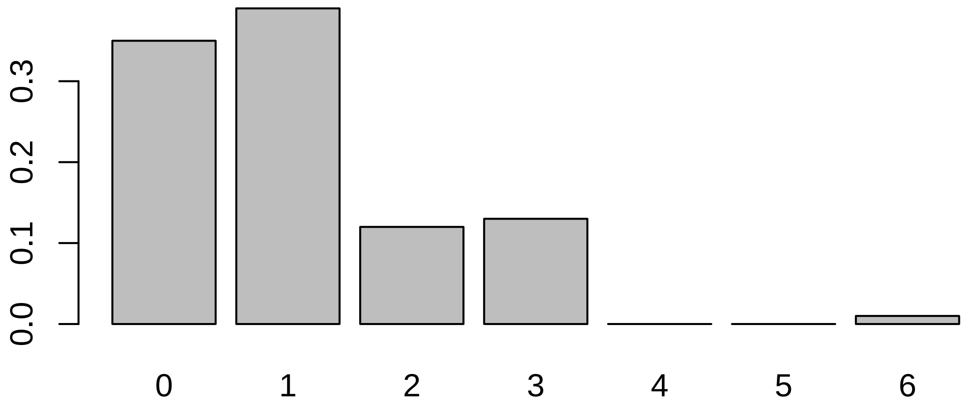


Figure 3.1: Bar plot of the observed surface flaw distribution.

```r
1 - pFlaws(4)

## [1] 0.01
```

3.1.3 *Discrete pseudorandom number generation*

To generate n pseudorandom numbers that follow a discrete distribution model for a random variable X, we can use a function such as the following:

```r
rFlaws <- function(n) {
    U <- runif(n)
    X <- numeric(n)
    x <- 0
    for (x in 0:6) {
        X[U >= pDiscreteDist(x)] <- x + 1
        x <- x+1
    }
    return(X)
}
```

Here, we have assumed that `pDiscreteDist(x)` is a function that returns the cumulative distribution function value at x: $P(X \leq x)$.

Example 3.7 *The following function shows how one might simulate large numbers of independent variates which follow the distribution of surface flaw counts obtained above:*

```
rFlaws <- function(n) {
    U <- runif(n)
    X <- numeric(n)
    x <- 0
    for (x in 0:6) {
        X[U >= pFlaws(x)] <- x + 1
        x <- x+1
    }
    return(X)
}
```

The following illustrates the use of the function to generate 10 values.

```
rFlaws(10)

##  [1] 1 0 0 1 3 0 2 0 3 1
```

Summarizing such data in a table is often appropriate

Example 3.8 *We simulate 100 values from the flaw distribution and tabulate the values as follows:*

```
table(rFlaws(100))

##
##  0  1  2  3  6
## 32 42 13 10  3
```

3.1.4 A model for the surface flaw distribution

The completely data-driven model and simulator produced earlier cannot predict counts of 4 or 5. This is not realistic. Models can provide more realism, at the expense of other kinds of inaccuracy. A possible model for the flaw distribution is

$d_F(x) = (1-(x/7)^2)^8/2.59676876439$, for $x = 0, 1, 2, \ldots, 6$ and 0, otherwise. The awkward number in the denominator of this expression ensures that the sum of all possible probabilities is 1.

The probability distribution can be coded into an R function:

```
dFlaw <- function(x) {
    (1-(x/7)^2)^8*(x >=0)*(x <= 6)/2.59676876439
}
```

Evaluating the function gives the probabilities of observing any of the possibilities, 0 through 6 flaws:

```
dFlaw(0:6)

## [1] 3.85094e-01 3.26534e-01 1.94849e-01 7.59413e-02 1.62971e-02
## [6] 1.27552e-03 9.45246e-06
```

The probabilities of 4 and 5 flaws are now small, but nonzero. The total probability is still

```
sum(dFlaw(0:6))

## [1] 1
```

Simulating from the distribution model

The following function be used to simulate random numbers from the flaw distribution model:

```
rFlaw <- function(n) {
    F <- function(f) sum(dFlaw(0:f))
    U <- runif(n)
    X <- numeric(n)
    for (j in 0:5) {
        X[U > F(j)] <- j + 1
    }
return(X)
}
```

In the above code, we have used the locally defined function `F` to calculate the cumulative distribution function for the flaw model. Tabulating the simulated values proceeds as before:

```
table(rFlaw(100000))/100000

##
##       0       1       2       3       4       5
## 0.38377 0.32836 0.19555 0.07570 0.01542 0.00120
```

3.2 Expected value

The expected value of a distribution is a single number which provides a partial summary of the distribution of a random variable X. It is also known as the mean and denoted as $E[X]$. The expected value is calculated according to

$$E[X] = \sum_{j=0}^{\infty} j d_X(j)$$

where $d_X(j) = P(X = j)$.

Example 3.9 *To find the expected value of the data-driven model for the counts of flaws, we can use*

```
j <- 0:6
sum(dFlaws(j)*j)

## [1] 1.08
```

Note that this matches the average of the data exactly: $(39 + 12(2) + 13(3) + 1(6))/100 = 1.08$.

Example 3.10 *To find the expected value of the alternative model for the counts of flaws, we proceed as follows:*

```
j <- 0:6
sum(dFlaw(j)*j)

## [1] 1.01568
```

Note that this no longer matches the average of the data but is not far off.

3.2.1 *How far is far?*

When measuring the distance between random quantities such as averages, it is advisable to consider the amount of variation in those quantities. If the distributions of the two random quantities do not overlap substantially, we could be fairly certain that the quantities differ. If there is substantial overlap, we must be open to the possibility that the quantities are equal.

Example 3.11 *We can compare the distributions of averages coming from each of the surface flaw models by use of simulation. For each distribution, we simulate a large number of samples of 100 counts, and we compute the average of each sample. We can then compare the distributions with side-by-side boxplots.*

We will store the averages from the original flaws model in `Averages1` *and the averages for the approximate flaws model in* `Averages2`*.*

```
N <- 1000 # Number of samples
Averages1 <- Averages2 <- numeric(N)
for (j in 1:N) {
    Averages1[j] <- mean(rFlaws(100))
    Averages2[j] <- mean(rFlaw(100))
}
boxplot(Averages1, Averages2)
```

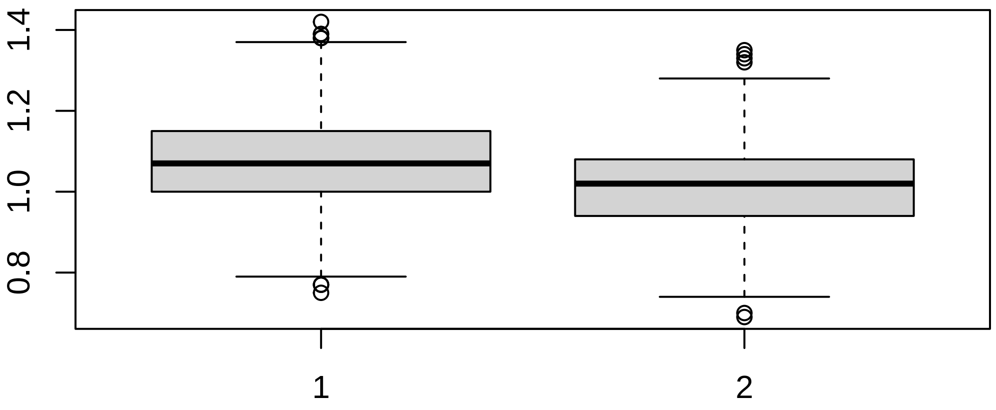


Figure 3.2: Side-by-side boxplots of the averages for the two surface flaw distribution models. The plot on the left is based on simulated samples coming from the data-based model. The plot on the right is based on the mathematical model.

Figure 3.2 shows the boxplots of the averages coming from the two types of models. There is a substantial overlap between the two distributions, and the medians are very close together, suggesting that the informal assertion of the preceding section was an accurate statement.

Randomly sampling from the data-based model to generate the sampling distribution of the average as was done in the above example is a form of nonparametric bootstrapping. Sampling from the second kind of model for the same purpose is sometimes referred to as parametric bootstrapping.

3.2.2 Variance and standard deviation

There are two mathematically equivalent formulas for the theoretical variance of a random variable X:

$$\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2.$$

The calculation of $E[X^2]$ is similar to the calculation of $E[X]$, for discrete random variables, with a j^2 in place of j:

$$E[X^2] = \sum_{j=0}^{\infty} j^2 d_X(j)$$

where $d_X(j) = P(X = j)$ is the probability distribution of X.

The standard deviation is defined as the square root of the variance.

$$S = \sqrt{\text{Var}(X)}.$$

Example 3.12 *To find the standard deviation of the alternative model for the counts of flaws, we can use*

```
j <- 0:6
V <- sum(dFlaw(j)*j^2) - (sum(dFlaw(j)*j))^2
SD <- sqrt(V)
SD

## [1] 1.02508
```

3.2.3 Standard error

Dividing by the square root of the sample size gives us the standard error which can be used to assess distance between an average and an expected value:

$$SEM = \frac{S}{\sqrt{n}}.$$

This formula is only appropriate when the measurements used to calculate the average are uncorrelated with each other.

The standard error can often provide a useful measure of distance. For the sample mean of independent measurements, the standard error is the standard deviation of a measurement divided by the square root of the sample size. Often, we would judge two quantities to be far apart, if the distance between them exceeds 2 standard errors.

Example 3.13 *The standard error of the simulated values from the*

```
SD/sqrt(100)   # standard error

## [1] 0.102508
```

Since the observed average and the alternative model mean differ by less than this amount, we would conclude that the two values are close together.

In the above example, we treated the average of the original sample as a fixed value, and we were interested in establishing whether averages from the flaw model would be different from this value. A more precise approach to this problem would be to calculate the standard error of the difference in the averages.

3.3 Special discrete random variable models

The following are the most commonly used discrete probability distributions.

- Bernoulli
- Binomial
- Geometric
- Poisson
- Negative Binomial

3.3.1 *Bernoulli random variables*

A Bernoulli trial is an experiment in which there are only 2 possible outcomes. For example, a light bulb may work or not work; these are the only possibilities. Each outcome (‘work’ or ‘not work’) has a probability associated with it; the sum of these two probabilities must be 1. Other possible outcome pairs are: (living, dying), (success, failure), (true, false), (0, 1), (-1, 1), (yes, no), (black, white), (go, stop) . . ..

Bernoulli experiments yield binary data.

3.3.2 *Simulating a Bernoulli random variable*

We could also think about outcomes that come from simulating a uniform random variable U on $[0, 1]$.

For example, the event that U is less than 0.2 is a possible outcome. It occurs with probability 0.2. It does not occur with probability 0.8. The outcome pair is ($U < 0.2, U \geq 0.2$). We can associate the event $U < 0.2$ with an event that we want to simulate.

```
set.seed(88832)  # use this to replicate the results below
```

Example 3.14 *Consider a student who guesses on a multiple choice test question which has 5 possible answers, of which exactly 1 is correct. The student may guess correctly with probability 0.2 and incorrectly with probability 0.8. We can simulate the correctness of the student on one question with a $U[0, 1]$ random variable. If the outcome is* `TRUE`*, the student guessed correctly; otherwise the student is incorrect.*

```
U <- runif(1)  # generate U[0,1] number
U

## [1] 0.712541

U < 0.2 # test U < 0.2 and simulate student's outcome

## [1] FALSE
```

In the preceding example, we could associate the values ‘1’ and ‘0’ with the outcomes from a Bernoulli trial. This defines the Bernoulli random variable: a random variable which takes the value 1 with probability p, and 0 with probability $1 - p$.

3.3.3 *Expected value of a Bernoulli random variable X*

The expected value of a Bernoulli random variable is p: $E[X] = p$. This follows from the fact that $X = 1$ with probability p and $X = 0$ with probability $1 - p$:

$$E[X] = 0 \times P(X = 0) + 1 \times P(X = 1) = 1p = p.$$

3.3.4 Variance of a Bernoulli random variable X

The variance of a random variable X can be calculated from the formula:

$$\text{Var}(X) = E[X^2] - (E[X])^2.$$

For a Bernoulli random variable,

$$E[X^2] = E[X] \text{ since } X^2 = X$$

which is true for any variable which can only take on values 0 or 1. Therefore, the variance of X is

$$E[X] - (E[X])^2 = p - p^2 = p(1-p).$$

The standard deviation is the square root of the variance: $\sqrt{p(1-p)}$.

In the above example, a student would expect to guess correctly 20% of the time, and the standard deviation is $\sqrt{.16} = .4$.

Binary data with a trend

Suppose there are 40 students in the class, and some study and some do not. Let s denote the number of hours that a student studies. A possible model for the probability of answering a question correctly might be

$$d(s) = .8 - .6\text{e}^{-s}$$

which gives the probability of 0.2 for no studying, and a probability of .8 for an unlimited amount of studying (there are other factors besides studying that influence a student's performance). We might model the number of hours of study for each student with a uniform distribution on $[0, 4]$. We can simulate such a class using

```
n <- 40
S <- runif(n, min = 0, max = 4)
S # number of study hours for each student

##  [1] 2.88560 0.70515 0.74770 2.88102 0.30833 0.57303 2.92813 1.33874
##  [9] 2.15108 0.83731 0.43262 3.49134 3.57069 0.47193 1.50513 1.24458
## [17] 2.95007 0.85778 1.00540 2.12898 1.26908 3.42819 2.77051 0.00868
## [25] 3.48061 2.52366 3.59945 3.19887 3.29401 1.09803 1.70981 1.80608
## [33] 1.21620 2.16889 0.15092 1.81191 3.94547 2.75876 1.96593 0.30341
```

```
U <- runif(n)
U < .8 - .6*exp(-S) #  results for the first question

##  [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## [12] FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE
## [23]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
## [34]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

Simulating the class performance on a 20 question test:

```
Ncorrect <- (U < .8 - .6*exp(-S)) # 1st question
for (i in 2:20) {
U <- runif(n)
Ncorrect <- Ncorrect + (U < .8 - .6*exp(-S))
}
table(Ncorrect)

## Ncorrect
##  4  5  6  8 10 11 12 13 14 15 16 17
##  1  1  2  1  2  2  5  6  6  3  5  6
```

Visualizing the trend

Figure 3.3 displays a scatter plot of the number of correct answers as a function of number of hours studied.

```
plot(Ncorrect ~ S, pch = 16, xlab = "Study Time (in hours)",
   ylab="Number Correct")
```

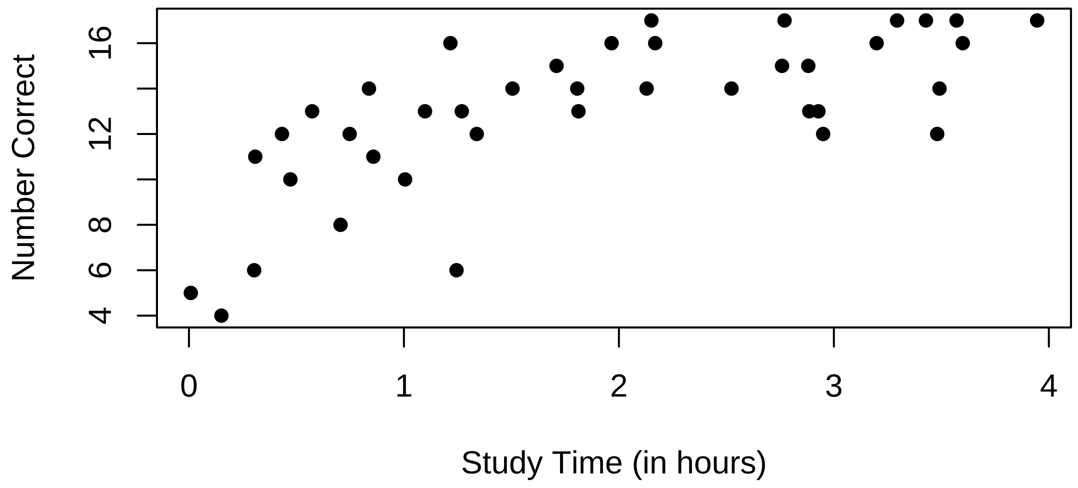


Figure 3.3: Scatter plot of simulated number of correct answers versus hours studied.

3.3.5 *Binomial random variables*

Let X denote the sum of m independent Bernoulli random variables, each having probability p. X is called a binomial random variable; it represents the number of 'successes' in m Bernoulli trials. A binomial random variable can take values in the set
$\{0, 1, 2, \ldots, m\}$.

Example 3.15 *Let's simulate the number of 6's tossed by someone rolling a die 30 times:*

```
set.seed(23207) # use this to obtain our output
m <- 30 # number of rolls
p <- 1/6 # probability of getting a 6
uniformNumbers <- runif(m) # 30 uniform numbers
NumberofSixes <- (uniformNumbers < p) # die rolls
sum(NumberofSixes)

## [1] 6
```

In this case, the number of 6's rolled is very near what we would expect in 30 *rolls. That is, we got* 6 *6's and expected* 5*. If we simulate another* 30 *rolls, we will get a different number as in the next example.*

Example 3.16
```
m <- 30 # number of rolls
p <- 1/6 # probability of getting a 6
uniformNumbers <- runif(m) # m uniform numbers
NumberofSixes <- (uniformNumbers < p) # die rolls
sum(NumberofSixes)

## [1] 8
```

This time we got more 6's than expected.

If we repeatedly simulate this process, we can visualize this binomial distribution with a table or bar chart as on the next slides.

```
m <- 30; N <- 500 # number of simulations
p <- 1/6 # probability of getting a 6
RandomBinomial <- numeric(N) # store 500 binomial numbers here
for (j in 1:N) { # use a for loop to repeat simulation N times
    uniformNumbers <- runif(m) # m uniform numbers
    NumberofSixes <- (uniformNumbers < p) # die rolls
    RandomBinomial[j] <- sum(NumberofSixes)
}
table(RandomBinomial)

## RandomBinomial
##  0  1  2  3  4  5  6  7  8  9 10 11 12
##  2 16 33 70 89 90 83 57 28 19  7  5  1
```

We convert the table to a bar plot as follows:

```
barplot(table(RandomBinomial))
```

Figure 3.4 displays the distribution. It shows that the probability of 5 is a bit higher than 4 or 6 and much higher than all other possibilities. Note that the distribution is almost bell-shaped, like the normal distribution. This is not a coincidence.

3.3.6 Calculating binomial probabilities

The probability of a binomial random variable X taking on any value can be computed using the `dbinom()` function.

```
dbinom(x, size, prob)
```

Here, `size` and `prob` are the binomial parameters m and p, respectively, while `x` denotes the number of 'successes'. The output from this function is the value of $P(X = x)$.

Example 3.17 *Compute the probability of getting* 5 *6's in 30 rolls of a fair die.*

```
dbinom(x = 5, size = 30, prob = 1/6)

## [1] 0.192108
```

Thus, $P(X = 5) = 0.1921$, when X is a binomial random variable with $m = 30$ and $p = \frac{1}{6}$.
Compute the probability of getting 4 heads in 6 tosses of a fair coin.

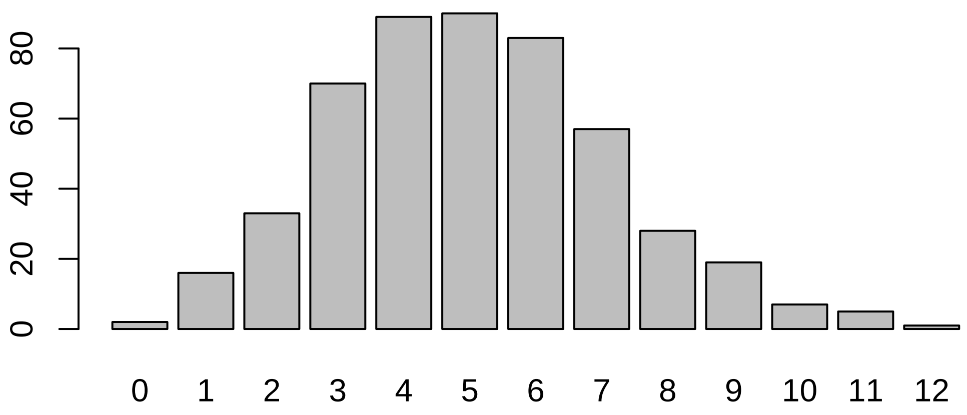


Figure 3.4: Bar plot exhibiting the distribution of a binomial random variable whose probability of success is 1/6.

```
dbinom(x = 4, size = 6, prob = 0.5)

## [1] 0.234375
```

Thus, $P(X = 4) = 0.234$*, when* X *is a binomial random variable with* $m = 6$ *and* $p = 0.5$*.*

Cumulative probabilities of the form $P(X \leq x)$ can be computed using `pbinom()`; this function takes the same arguments as `dbinom()`. For example, we can calculate $P(X \leq 4)$ where X is the number of heads obtained in 6 tosses of a fair coin as:

```
pbinom(4, 6, 0.5)

## [1] 0.890625
```

3.3.7 Binomial pseudorandom numbers

The `rbinom()` function can be used to generate binomial pseudorandom numbers.

```
rbinom(n, size, prob)
```

Here, `size` and `prob` are the binomial parameters, while `n` is the number of variates generated.

Example 3.18 *This example demonstrates the essential idea of a control chart which is a very useful data analytic tool in many industries.*

Suppose 10% of the windshields produced on an assembly line are defective, and suppose 15 windshields are produced each hour. Each windshield is independent of all other windshields. This process is judged to be out of control when more than 4 defective windshields are produced in any single hour.

Simulate the number of defective windshields produced for each hour over a 24-hour period, and determine if any process should have been judged out of control at any point in that simulation run.

Example 3.19 *Since 15 windshields are produced each hour and each windshield has a 0.1 probability of being defective, independent of the state of the other windshields, the number of defectives produced in one hour is a binomial random variable with* $m = 15$ *and* $p = 0.1$.

To simulate the number of defectives for each hour in a 24-hour period, we need to generate 24 binomial random numbers. We then identify all instances in which the number of defectives exceeds 5.

One such simulation run is:

```
defectives <- rbinom(24, 15, 0.1)
defectives

##  [1] 1 3 2 1 1 2 2 4 1 1 2 0 2 4 1 1 4 1 1 0 1 1 2 3
```

```
any(defectives > 5) # check if any defective counts exceed 5

## [1] FALSE
```

The defective counts for each of the 24 hours are below the cut-off (5) so there is no indication that the process is out of control.

3.3.8 Simulating geometric random variables

We saw that a binomial random variable is defined as the sum of m independent Bernoulli random variables. Now, we will define a geometric random variable as the *number* of Bernoulli random variables that must be generated before the first 1 appears.

Example 3.20 *Suppose we toss a coin repeatedly and want to count the number of heads (0) until we toss the first tail.*

If we simulate 5 *coin tosses, we obtain*

```
p <- 0.5; # probability of a tail
runif(5) < p

## [1] FALSE FALSE  TRUE FALSE  TRUE
```

The first 2 *tosses are heads and the third toss is a tail, so in this example, the geometric random number would be* 2.

The third, fourth and fifth tosses weren't needed, so we would be better off using a `while()` *loop here.*

Example 3.21
```
set.seed(123488) # use this seed to get our result
p <- 0.5; # probability of a tossing a tail
G <- 0 # eventual value of geometric random variable
U <- runif(1) # first coin toss
IsItaTail <- (U <= p) # this will be TRUE when we toss a tail
while (IsItaTail == FALSE) {
    G <- G + 1 # add one to G until IsItaTail is TRUE
    U <- runif(1)
    IsItaTail <- (U <= p)
}
G

## [1] 1
```

Example 3.22 *This time simulate the number of independent die rolls until rolling a 6.*

```
p <- 1/6; # probability of a rolling a 6
G <- 0 # eventual value of geometric random variable
U <- runif(1) # first die roll
IsIt6 <- (U < p) # TRUE if we roll a 6
while (IsIt6 == FALSE) {
    G <- G + 1 # add one to G until IsIt6 is TRUE
    U <- runif(1)
    IsIt6 <- (U < p)
}
G

## [1] 12
```

Example 3.23 *This time simulate the number of independent 2-dice rolls until rolling a 12.*

```
p <- 1/6; # probability of a rolling a 6
U <- runif(2) # first 2 dice rolls
G <- 0 # eventual value of geometric random variable
IsIt12 <- (U[1]<1/6)&(U[2]<1/6) # TRUE if both dice are 6's
while (IsIt12 == FALSE) {
    G <- G + 1 # add one to G until both dice are 6's
    U <- runif(2)
    IsIt12 <- (U[1]<1/6)&(U[2]<1/6) #
}
G

## [1] 4
```

We can also use the `rgeom()` *function to simulate these numbers. For example, to simulate the number of die rolls until the first 6, use*

```
G <- rgeom(1, p = 1/6)
G

## [1] 3
```

To simulate the number of 2-dice rolls until the first 12 (an event with probability 1/36), use

```
G <- rgeom(1, p = 1/36)
G

## [1] 65
```

Let's look at the distribution of a geometric random variable where $p = 1/2$ *by simulating 500 values and tabulating the result:*

```
N <- 500;
G <- rgeom(N, p = 1/2)
table(G)

## G
##   0   1   2   3   4   5   6   7   9
## 252 134  53  32  13   9   3   3   1
```

The bar plot is displayed in Figure 3.5.

```
barplot(table(G))
```

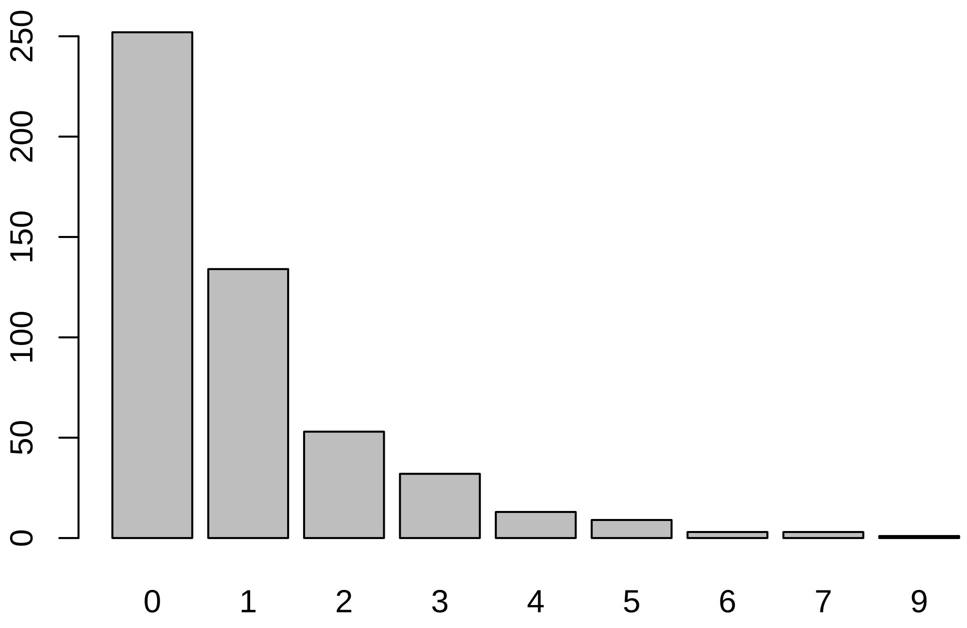


Figure 3.5: Bar plot of simulated geometric random variables.

From the plot, we see that the probability of a 0 is higher than for a 1 which is a higher than for a 2 and so on.

3.3.9 Calculating geometric probabilities

The probability of a geometric random variable X taking on any value can be computed using the `dgeom()` function.

```
dgeom(x, prob)
```

Here, `prob` is the parameter p, while `x` denotes the number of trials before the first 'success'. The output from this function is the value of $P(X = x)$.

Example 3.24 *Compute the probability that it will take 5 die rolls before obtaining the first 6.*

```
dgeom(x = 5, prob = 1/6)

## [1] 0.0669796
```

Thus, $P(X = 5) = 0.067$*, when* X *is a geometric random variable with* $p = \frac{1}{6}$.

Compute the probability that it will take 5 or fewer die rolls before obtaining the first 6. (Use the `pgeom()` *function for this.)*

```
pgeom(5, prob = 1/6)

## [1] 0.665102
```

Thus, $P(X \leq 5) = 0.6651$*, when* X *is a geometric random variable with* $p = \frac{1}{6}$.

3.3.10 The probability of a 100-year disaster

The geometric distribution is the simplest of all models that can be used to predict the occurrence of a disaster, such as a flood. If the probability of a disaster in a given year is .01, how long would we expect to wait for the event?

If we simulate a large number of geometric random variables with $p = .01$, we can visual the distribution of the waiting time:

```
W <- rgeom(500, p = .01)
mean(W) # this gives us the average waiting time

## [1] 97
```

This is what is meant by a 100-year event. But note that the 100-year event could happen pretty soon:

```
hist(W)
```

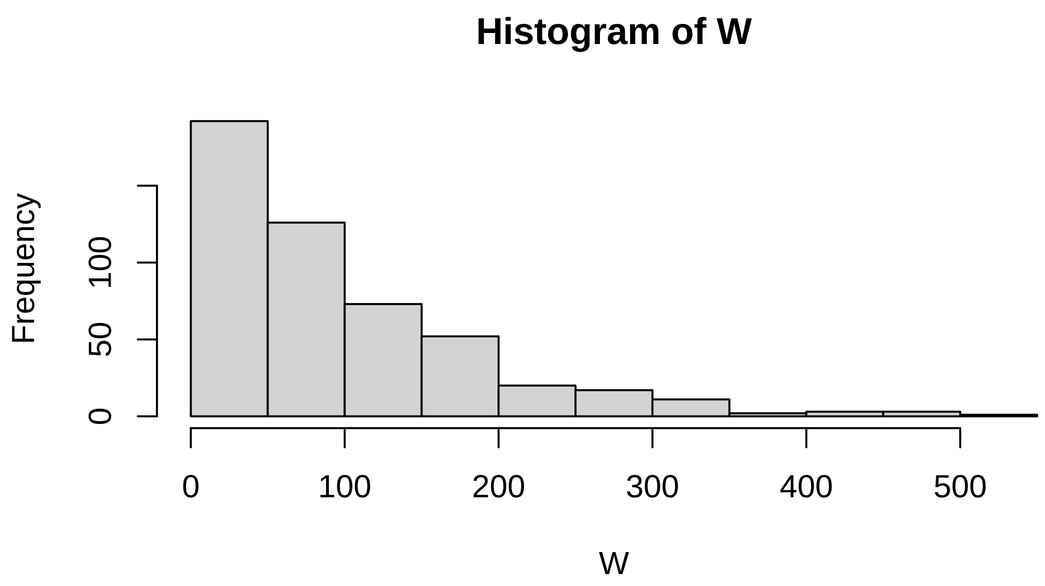


Figure 3.6: Histogram of simulated geometric random variables representing the waiting times until a 100-year event.

Figure 3.6 displays a histogram of the simulated waiting times. The probability of the event occurring within the next 25 years is

```
pgeom(25, p = .01)

## [1] 0.229957
```

Example 3.25 *Using the binomial distribution, calculate the probability that 2 such disasters could occur in the same 100 year period.*

The probability of 1 or fewer disasters in 100 years is

```
pbinom(1, 100, p=.01)

## [1] 0.735762
```

so we can subtract this from 1 to get the required probability:

```
1-pbinom(1, 100, p=.01)

## [1] 0.264238
```

3.3.11 The expected value of a geometric random variable

For the geometric random variable X with $P(X = j) = p(1-p)^j$, for $j = 1, 2, \ldots$, it can be shown using arguments involving geometric series, that

$$E[X] = \frac{1-p}{p}.$$

Note that if the variable is defined so that $P(X = j) = p(1-p)^{j-1}$, then

$$E[X] = \frac{1}{p}.$$

3.4 Poisson random variables

The Poisson distribution is the limit of a sequence of binomial distributions with parameters n and p_n, where n is increasing to infinity, and p_n is decreasing to 0, but where the expected value (or mean) np_n converges to a constant λ.

The variance $np_n(1-p_n)$ converges to this same constant. Thus, the mean and variance of a Poisson random variable are both equal to λ. This parameter is sometimes referred to as a *rate*.

3.4.1 Applications of Poisson random variables

Poisson random variables arise in a number of different ways. They are often used as a crude model for count data.

Examples of count data are the numbers of earthquakes in a region in a given year, or the number of individuals who arrive at a bank teller in a given hour. The limit comes from dividing the time period into n independent intervals, on which the count is either 0 or 1. The Poisson random variable is the total count.

3.4.2 Distribution of Poisson random variables

The possible values that a Poisson random variable X could take are the non-negative integers $\{0, 1, 2, \ldots\}$.

The probability of taking on any of these values is

$$P(X = x) = \frac{e^{-x}\lambda^x}{x!}, \quad x = 0, 1, 2, \ldots.$$

Calculation of Poisson probabilities

The Poisson probabilities can be evaluated using the `dpois()` function.

```
dpois(x, lambda)
```

Here, `lambda` is the Poisson rate parameter, while `x` is the number of Poisson events. The output from the function is the value of $P(X = x)$.

Example 3.26 *The average number of arrivals per minute at an automatic bank teller is 0.5. Assuming the number arriving follows a Poisson distribution, the probability of 3 arrivals in the next minute is*

```
dpois(x = 3, lambda = 0.5)

## [1] 0.0126361
```

Therefore, $P(X = 3) = 0.0126$*, if* X *is Poisson random variable with mean* 0.5.

Poisson pseudorandom numbers

The following function shows how one might simulate large numbers of independent Poisson variates:

```
rPois <- function(n, lambda) {
    U <- runif(n)
    X <- numeric(n)
    x <- 0
    while (max(U) > ppois(x, lambda)) {
        X[U >= ppois(x, lambda)] <- x + 1
        x <- x+1
    }
    return(X)
}
```

Example 3.27
```
n <- 100000; rate <- 3
X <- rPois(n, rate)
X[1:5] # the first 5 variates

## [1] 1 3 4 2 5
```

Tabular summary of the Poisson counts:

```
table(X)

## X
##     0     1     2     3     4     5     6     7     8     9    10
##  5007 14946 22474 22335 16755 10140  4962  2135   887   243    82
##    11    12    13
##    26     7     1
```

We can compare the simulated probabilities with the theoretical probabilities as follows:

```
table(X)/n - dpois(0:max(X), rate)

## X
##             0             1             2             3             4
##  2.82932e-04  9.87949e-05  6.98192e-04 -6.91808e-04 -4.81356e-04
##             5             6             7             8             9
##  5.81187e-04 -7.89407e-04 -2.54031e-04  7.68488e-04 -2.70504e-04
##            10            11            12            13
##  9.84882e-06  3.90497e-05  1.47624e-05 -2.74713e-06
```

We can also generate Poisson random numbers using the rpois() function.

```
rpois(n, lambda)
```

The parameter `n` is the number of variates produced, and `lambda` is as above.

3.4.3 *Poisson probabilities and quantiles*

Cumulative probabilities of the form $P(X \leq x)$ can be calculated using `ppois()`, and Poisson quantiles can be computed using `qpois()`.

Example 3.28 *Suppose traffic accidents occur at an intersection with a mean rate of 3.7 per year. Simulate the annual number of accidents for a 10-year period, assuming a Poisson model.*

```
rpois(10, 3.7)

##  [1] 2 4 3 8 3 7 4 0 1 7
```

3.5 Poisson processes

A Poisson process is a simple model of the collection of events that occur during an interval of time. A way of thinking about a Poisson process is to think of a random collection of points on a line or in the plane (or in higher dimensions, if necessary).

The homogeneous Poisson process has the following properties:

1. The distribution of the number of points in a set is Poisson with rate proportional to the size of the set.
2. The numbers of points in non-overlapping sets are independent of each other.

In particular, for a Poisson process with rate λ the number of points on an interval $[0, T]$ is Poisson distributed with mean λT.

One way to simulate a Poisson process is as follows.

1. Generate N as a Poisson pseudorandom number with parameter λT.

2. Generate N independent uniform pseudorandom numbers on the interval $[0, T]$.

Example 3.29 *Simulate points of a homogeneous Poisson process having rate 1.5 on the interval* $[0, 10]$.

```
N <- rpois(1, 1.5 * 10)
P <- runif(N, max = 10)
sort(P)

##  [1] 0.0536052 1.0879438 2.1613969 2.3459146 2.3776094 2.7841103
##  [7] 3.7293372 5.0788597 6.0528535 6.7569535 6.8712531 7.8808272
```

3.6 Summary

We can generate the building blocks for discrete simulation using uniform random numbers. Bernoulli random variables can be generated from uniforms. Binomial random variables are sums of independent Bernoullis and are a basic model for counting defectives. Poisson random variables are a basic model for counting defects. Negative binomial variables are sometimes useful as a more accurate model than the Poisson. (Geometric random variables are a special case.)

Exercises

1. Crates containing a large number of batteries are monitored by randomly selecting 20 from each crate and running an accelerated life test. If 2.5 percent of the batteries are defective under normal conditions, identify a model for the number of defective batteries in a given sample.
2. Identify an appropriate alternative model for the refrigerator surface flaw data.
3. Write R code to calculate the probability that a binomial random with parameters $n = 50000$ and $p = .5$ takes the value 25000 using
   (a) any function you wish.
   (b) only the functions `sum()`, `log()` and `exp()`.
4. Suppose k is a positive integer, and consider the discrete distribution
$$p_k(x) = \frac{2(k+1-x)}{k(k+1)}, \text{ for } x = 1, 2, \ldots, k$$
   and $p_k(x) = 0$, otherwise.
   (a) Write an R function which takes a numeric vector `x` and an integer `k` as input and returns a vector with the values of $p_k(x)$.
   (b) Verify that the sum of the probabilities is 1, when `k` is 7.
   (c) Construct an R function which takes `n` and `k` as input and returns a vector of pseudorandom numbers following the distribution $p_k(x)$.
   (d) Use the fact that $\sum_{j=1}^{k} j^2 = k(2k+1)(k+1)/6$ to find the expected value of X when X follows the distribution p_k.
   (e) Simulate 100000 values from the p_k distribution when $k = 10$, compute their average and compare with the expected value.
   (f) Suppose k independent values of X, coming from the distribution p_k, are observed. Let Y be the number of occurrences of the value k. What is the probability distribution of Y?
   (g) Suppose independent values of x are generated from the distribution p_k until the first occurrence of the value k. Let W denote the total number of values generated. What is the distribution of W?
   (h) Suppose independent values of x are generated from the distribution p_k until the kth occurrence of the value k. Let W denote the total number of values generated. What is the distribution of W?
   (i) Write an R function which takes as input `n` and `k` and returns n independent values from the distribution of W.

4

Modelling more than one discrete random variable

Until now, our focus has mostly been on the behaviour of a single random variable, using simulation and modelling tools to understand the visualize and describe univariate distributions. Now, we consider multivariate distributions, where several random variables are possibly interacting with each other. Our principle interest is in finding models for some of the patterns that relate two or more variables to each other.

4.1 Joint probability distributions

We focus on two discrete random variables initially. Extensions to higher dimensions are relatively straightforward, but our treatment, later on, will be less detailed.

4.1.1 Bivariate Bernoulli distribution

In order for valid probabilities to be computed, it is necessary that the probabilities be nonnegative, and its overall sum should be 1.

```
# Bivariate Bernoulli

dBer2 <- function(x, dprobs) {
    # x is a 2-column matrix
    # dprobs is a 4-vector of probabilities of each outcome
    probs[x%*%(2:1) + 1]
}

pBer2 <- function(x, pprobs) {
    # x is a 2-column matrix
    # pprobs is a 4-vector of cumulative probabilities
    #     pprobs <- c(dprobs[1], sum(dprobs[1:2]), sum(dprobs[c(1, 3)]), 1)
    pprobs[x%*%(2:1) + 1]
}
```

Crude simulation

Crude simulator:

```
rBer2 <- function(n, dprobs) {
    pprobs <- cumsum(dprobs) # (0, 0), (0, 1), (1, 0), (1, 1)
    U <- runif(n)
    X <- numeric(n) + 1
    for (x in 1:3) {
        X[U > pprobs[x]] <- x + 1
    }
    B1 <- factor(X); levels(B1) <- c(0, 0, 1, 1)
    B2 <- factor(X); levels(B2) <- c(0, 1, 0, 1)
    B1 <- as.numeric(as.character(B1))
    B2 <- as.numeric(as.character(B2))
    B <- cbind(B1, B2)
    return(B)
}
```

```
B <- rBer2(50, c(.1, .3, .4, .2))
B[1:5,]  # first five outcomes

##      B1 B2
## [1,]  1  1
## [2,]  1  0
## [3,]  1  0
## [4,]  1  0
## [5,]  0  1
```

```
table(paste(B[,1], B[,2], sep=','))

##
## 0,0 0,1 1,0 1,1
##   3  15  20  12
```

Gibbs sampling

Explanation.

Gibbs sampling is explained in considerable detail by Cappé et al. (2005).

Exercises

1. Suppose p_1, p_2, p_3 and p_4 are nonnegative real numbers that sum to 1, and X_0 and X_1 are random variables where
$$P(X_0 = 0, X_1 = 0) = p_1$$
$$P(X_0 = 0, X_1 = 1) = p_2$$
$$P(X_0 = 1, X_1 = 0) = p_3$$
and
$$P(X_0 = 1, X_1 = 1) = p_4$$
   (a) Show that $P(X_1 = 0) = p_1 + p_2$ and $P(X_2 = 0) = p_1 + p_3$. Thus, conclude that X_1 and X_2 are Bernoulli random variables.
   (b) Show that $P(X_2 = 0|X_1 = 0) = \frac{p_1}{p_1+p_2}$. Find analogous expressions for $P(X_2 = 0|X_1 = 1)$, $P(X_1 = 0|X_2 = 0)$ and $P(X_1 = 0|X_2 = 1)$.
   (c) Under what condition on the numbers $p_1, \ldots, p_4$ are X_1 and X_2 independent?

5
Discrete Time Markov Chains

We begin with a very simple example that illustrates many of the concepts that arise when studying and simulating from Markov chain models. Figure 5.1 represents the floor plan for a simple maze, consisting of four adjoining compartments, in which a mouse may randomly wander around. Some of the compartments are separated by walls that the mouse cannot pass through, and others such as compartments 1 and 2 are linked by a passageway, allowing the mouse to be able to pass back and forth. Similar passageways allow transition between compartments 2 and 3, and between compartments 3 and 4.

Even though the mouse may wait in a compartment for an arbitrary amount of time, we keep track only of the ordering of the transitions, and we do not worry about the amount of time that transpires between moves. We will often use the symbol X_n to denote the location or state of the mouse at time n. For example, if the mouse starts in compartment 1, and enters compartment 2, then compartment 3, back to 2, back to 3, then 4, and back to 3, and so on, we write this as

$$X_0 = 1, X_1 = 2, X_2 = 3, X_3 = 2, X_4 = 3, X_5 = 4, X_6 = 3, \ldots .$$

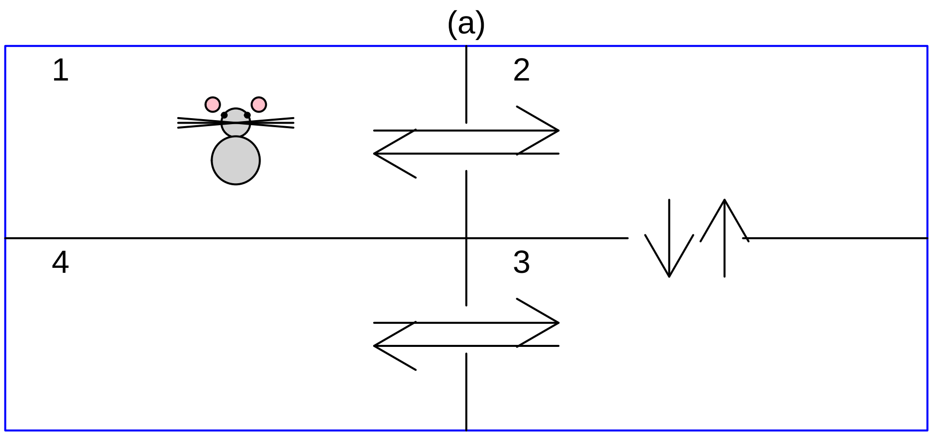


Figure 5.1: A four-compartment maze in which a mouse can wander from compartment to compartment.

If the mouse wanders aimlessly, we might attach probabilities to the transitions between compartments in the maze. We assume complete randomness. In other words, the mouse is just moving randomly around, stopped only by the walls between the compartments. Alternatively, the mouse might favour one of the compartments over another, perhaps because of the location of food or a particular odor.

Under this model, we can calculate the conditional probability that the mouse will enter a particular compartment, given that the mouse is currently located in another compartment. This conditional probability is known as transition probability. Note that we are not concerned with the duration of the mouse's stay within a compartment.

To calculate the conditional probability that the mouse would enter compartment 2, given that the mouse is currently in compartment 1, we observe that there is no other possible transition for the mouse. If the mouse is moving around at random, eventually, the mouse will find the passageway to compartment 2, and we would conclude that the transition probability from compartment 1 to compartment 2 is exactly 1.0.

Once in compartment 2, the mouse could possibly return to compartment 1, but it is equally likely to find the passageway to compartment 3. Therefore, the transition probabilities from compartment 2 to compartments 1 and compartments 3 are both 0.5. It is not possible for the mouse to travel directly to compartment 4, so this transition probability is 0. The other nonzero transition probabilities are noted in red in Figure 5.2.

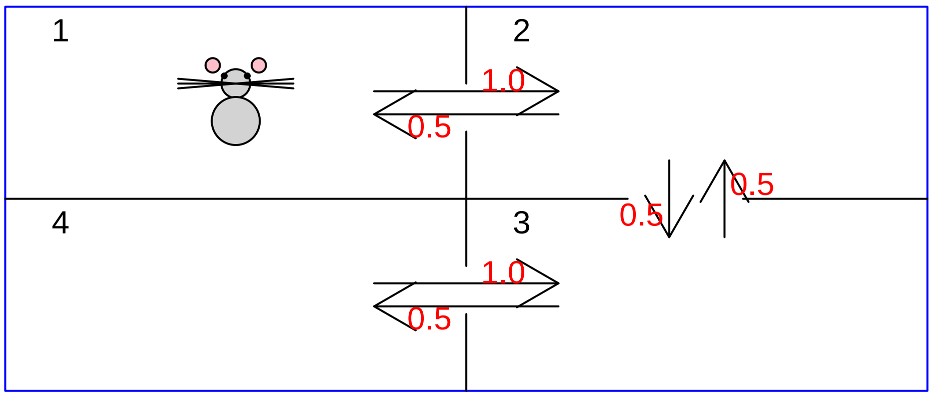


Figure 5.2: The mouse maze with specified probabilities of transition from compartment to compartment.

5.1 Some terminology

The possible values that an element of a Markov chain can take are referred to as states. The state space (S) is the set of all possible states that a Markov chain can visit, e.g. for the mouse maze example, the state space is $\{1, 2, 3, 4\}$.

Definition: The sequence of random variables $X_1, X_2, X_3, \ldots,$ is called a Markov chain if

$$P(X_n = j_n | X_{n-1} = j_{n-1}, X_{n-2} = j_{n-2}, \ldots)$$
$$= P(X_n = j_n | X_{n-1} = j_{n-1})$$

where $j_n, j_{n-1}, j_{n-2}, \ldots$ are elements of S.

The equation above is a form of the Markov Property. The sequence of mouse movements in the maze is an example of discrete time Markov chain. **Definition**: The transition probabilities can be organized systematically into an array, called the transition matrix, noting that impossible transitions have probability 0. The matrix P is called a transition matrix with (i, j)th entry

$$p_{ij} = P(X_n = j | X_{n-1} = i)$$

p_{ij} is the transition probability from state i to state j. (We are assuming that p_{ij} does not depend on n.)

Example 5.1 *The transition matrix for the mouse maze model is a 4×4 matrix. The first row of the transition matrix lists the transition probabilities from the first compartment to all other compartments. We assume that the mouse is not staying in compartment 1, so the first row consists of the values: 0, 1, 0, 0.*

To find the second row, we note that the transition probabilities out of compartment 2 are: 0.5, 0, 0.5, 0. For the third row, the transition probabilities from compartment 3 are: 0, 0.5, 0, 0.5. Finally, once the mouse reaches the fourth row, there is only way out, so the fourth row consists of the transition probabilities: 0, 0, 1, 0. So P for this model is:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.1}$$

where entry (i, j) represents the probability of transition into compartment j, given that the mouse was in state i;

5.2 Simulating from the Mouse Movement Model

X_n denotes the compartment holding the mouse at time n. Since the mouse starts in the first compartment, $X_0 = 1$. At the first transition, the mouse enters the second compartment: $X_1 = 2$ with probability 1. At the second transition, the mouse can enter compartment 1 with probability 0.5 or compartment 3 with probability 0.5 as seen in Figure 5.3.

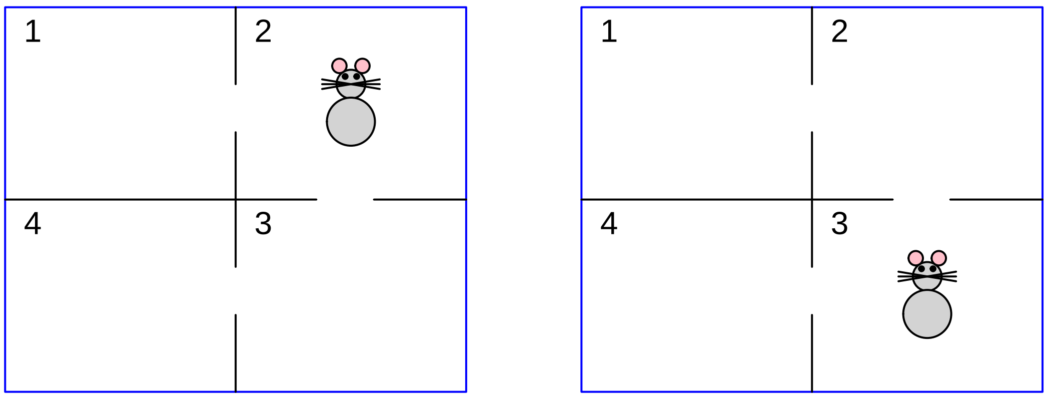


Figure 5.3: The first two transitions. The mouse has moved from compartment 1 to compartment 2 in the left panel and then on to compartment 3 in the right panel.

To simulate the second transition, we need to generate a pseudorandom number which we will call U_2. If $U_2 < 0.5$, we will make a rule that the mouse should enter compartment 1. Otherwise, it will enter compartment 3. If we obtain the value $U_2 = 0.61$, then the mouse enters the 3rd compartment: $X_2 = 3$.

For the succeeding transitions, we continue generating uniform random variates, U, on the interval $[0, 1]$, choosing to move to the compartment labelled with the lower value if $U < 0.5$, and choosing to move to the higher valued compartment if $U \geq 0.5$, unless the mouse is already in compartment 4.

A possible sequence of U values is: $U_3 = 0.34$, $U_4 = 0.88$, $U_5 = 0.52$. Then the mouse locations are: $X_3 = 2$, $X_4 = 3$, $X_5 = 4$. The mouse must re-enter compartment 3 with probability 1, so $X_6 = 3$, Continuing in this way, we obtain the sequence of values of $\{X_1, X_2, X_3, \ldots\}$, which is an example of a Markov chain realization.

5.2.1 Using R to simulate from Markov chains model

Automating the Markov chain simulation process with R can be done in several ways. A simple way to simulate values of X_j given the value of X_{j-1} is to use the `sample()` function.

Example 5.2 *One application of* `sample()` *is to simple random sampling from a population. Here, we show how to take a random sample, without replacement, of size 10 from a population consisting of the first 20000 numbers.*

```
        N <- 20000 # population size
        n <- 10 # sample size
        sample(1:N, size = n, replace = FALSE)
```

The `sample()` function also takes a `prob` argument which specifies different probability weights for each possible value. This allows us to sample from discrete probability distributions of any type.

Example 5.3 *Consider the problem of simulating from a discrete probability distribution*

$$P(X = 1) = .1, P(X = 2) = .2, P(X = 3) = .1, P(X = 4) = .4, P(X = 5) = .2$$

A sample of 6 values can be obtained from the distribution using

```
sample(1:5, size = 6, replace = TRUE,
prob = c(.1, .2, .1, .4, .2))

## [1] 2 3 2 3 5 4
```

When simulating values in a Markov chain, we are simulating from probability distributions defined by the rows of the transition matrix.

Example 5.4 *We will simulate 100 transitions from the following transition matrix for a 4-state Markov chain, starting from $X_0 = 1$.*

$$P = \begin{bmatrix} 0.1 & 0.2 & 0.5 & 0.2 \\ 0.5 & 0.2 & 0.3 & 0 \\ 0 & 0.4 & 0.1 & 0.5 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This is coded in R as:

```
P <- matrix(c(
         0.1, 0.2, 0.5, 0.2,
        0.5, 0.2, 0.3, 0.0,
        0.0, 0.4, 0.1, 0.5,
        0.0, 0.0, 1.0, 0.0
        ), nrow = 4,  byrow = TRUE)
```

In our simulation, we use the vector object `X` *to contain the $N = 100$ values $X_1, X_2, \ldots, X_N$, with the initial value $X_0 = 1$.*

```
        N <- 100 # number of transitions
        X <- numeric(N)# initializing the chain
        current.state <- 1 # initial state X_0 = 1
```

At each transition, the probability distribution for the next transition is specified according to the row of the transition matrix corresponding to the current state. The `sample` *function is then chooses the next state from the set $\{1, 2, 3, 4\}$ according to the specified distribution, according to the for loop below.*

```
        for (t in 1:N) {
        current.state <- sample(1:4,  size = 1, prob = P[current.state, ])
        X[t] <- current.state
        }
```

The first ten values of the simulation run were

```
        X[1:10]

##  [1] 2 3 4 3 4 3 2 3 4 3
```

It is useful to tabulate the results so that we can see the frequency of visits to the various states:

```
        table(X) # this counts the number of visits to each state

## X
##  1  2  3  4
## 11 23 44 22
```

We see that state 3 is visited more often than the other states, and state 1 is visited the least often.

We will run code similar to what was used in the previous example many times. Therefore, it will be convenient to call the following function instead of writing the same kind of code repeatedly.

```
rMC <- function(N = 1, initial.state = 1, P) {
    X <- numeric(N) # initializing
    current.state <- initial.state
    for (n in 1:N) {
        current.state <- sample(1:ncol(P), size = 1, prob = P[current.state, ])
        X[n] <- current.state
    }
    return(X)
}
```

Example 5.5 *We can simulate from the mouse maze model in the same way using the transition matrix given at (5.1). This time we will simulate 100000 values of the Markov chain* $(X_0, X_1, \ldots X_{99999})$*. We can compute the relative frequency of the number of visits by simply dividing by the total number of transitions.*

```
        P <- matrix(c(
        0.0, 1.0, 0.0, 0.0,
        0.5, 0.0, 0.5, 0.0,
        0.0, 0.5, 0.0, 0.5,
        0.0, 0.0, 1.0, 0.0
        ), nrow = 4, byrow = TRUE)
```

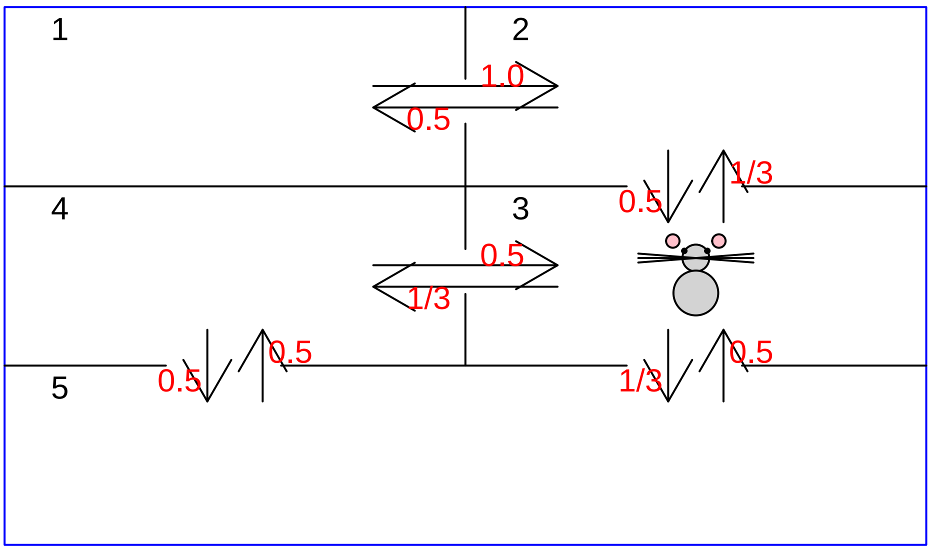
```
N <- 100000
X <- rMC(N = N, initial.state = 1, P = P)
table(X)/N  # relative frequency distribution

## X
##       1       2       3       4
## 0.16550 0.33224 0.33450 0.16776
```

Observe this time that the probability of visiting states 1 and 4 is essentially half of the probability of visiting states 2 and 3. Can you see why this might be true?

5.2.2 A more complicated maze

Another maze is pictured below. This one contains a fifth compartment which can be accessed from the third and fourth compartments.



The transition matrix for this example, assuming complete randomness is

$$P_5 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 & 0 \end{bmatrix}$$

Simulating a large number of transitions of this Markov chain runs in exactly the same way as before, but with the modified transition matrix.

```
P5 <- matrix(c(0, 1, 0, 0, 0,    0.5, 0, 0.5, 0, 0,
               0, 1/3, 0, 1/3, 1/3,    0, 0, 0.5, 0, 0.5,
               0, 0, 0.5, 0.5, 0), nrow = 5, byrow = TRUE)
```

The simulation runs as follows, with 5 states instead of 4.

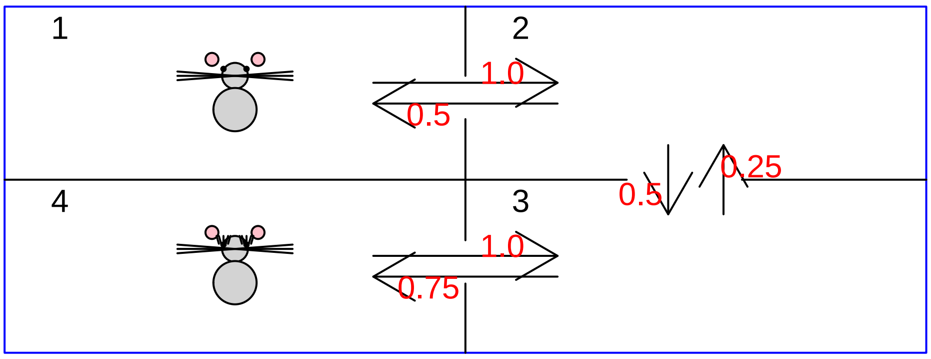
```
X <- rMC(N = N, initial.state = 1, P = P)
table(X)/N  # relative frequency distribution

## X
##       1       2       3       4
## 0.16915 0.33498 0.33085 0.16502
```

5.2.3 A mouse movement model - including some structure

The transition probabilities for a mouse maze experiment can be altered, possibly by including the odor of another mouse, usually of the opposite sex. In this case, the odor of a female mouse in the fourth compartment might induce an attraction to the male mouse. Once in compartment three, the odor increases the probability that the mouse enters compartment four to 0.75, while decreasing the probability of moving to compartment two to 0.25.

Transition Probabilities:



Transition Matrix:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.25 & 0 & 0.75 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Simulating the more structured model in R:

```
P <- matrix(c(0, 1, 0, 0,     0.5, 0, 0.5, 0,
              0, 0.25, 0, 0.75,     0, 0, 1, 0), nrow = 4,
              byrow = TRUE)  # P is the transition matrix
current.state <- 1
for (t in 1:Ntransitions) {
   current.state <- sample(1:4, size = 1, prob = P[current.state, ])
   location[t] <- current.state
}
table(location)
```

```
table(location)

## location
##     1     2     3     4
##  9928 19673 40072 30327

    # the odor in compartment 4 is attractive
```

The odor in compartment 4 is attractive, so the mouse makes more visits to that compartment than he would have when the odor is not present.

```
N <- 100000 # number of mouse moves
P <- matrix(c(0, 1, 0, 0,
0.5, 0, 0.5, 0,
0, 0.25, 0, 0.75,
0, 0, 1, 0), nrow = 4,
byrow = TRUE)  # P is the transition matrix
location <- numeric(N)
current.state <- 1
for (t in 1:N) {
```

```
current.state <- sample(1:4,
size = 1, prob = P[current.state, ])
location[t] <- current.state
}
```

```
table(location) # the odor in compartment 4 is attractive
```

If we simulate this Markov chain for a long time, we will find that the proportions of visits to each location follows a specific distribution. In the null model with four compartments, this stationary distribution is $1/6, 1/3, 1/3, 1/6$. In the model with the odor, the stationary distribution is different. (We will find out how to calculate it later.)

5.2.4 *Probability calculations*

$P(X_n = j|X_0 = i)$ is the (i, j)th element of the matrix P^n.

Example 5.6
```
P2 <- P%*%P
P2

##        [,1]  [,2]  [,3]  [,4]
## [1,] 0.500 0.000 0.500 0.000
## [2,] 0.000 0.625 0.000 0.375
## [3,] 0.125 0.000 0.875 0.000
## [4,] 0.000 0.250 0.000 0.750
```

For example, the probability of returning to compartment 1 after 2 transitions is 0.5. The probability of reaching compartment 4 in 2 transitions, if starting in compartment 2, is 0.375.

Define

$$x^{\{n\}} = [P(X_n = 1)\ P(X_n = 2)\ \ldots\ P(X_n = m)]$$

This vector is called the nth state vector of the Markov chain.

It specifies the probability distribution of X_n.

The sum of the entries of $x^{\{n\}}$ must always be one. $x^{\{0\}}$ denotes the distribution of the initial state X_0.

For the mouse example, if the mouse starts in compartment 1, $x^{\{0\}} = [1, 0, 0, 0]$.

If the mouse starts in a randomly selected compartment,
$x^{\{0\}} = [.25, .25, .25, .25]$.

$$x^{\{n\}} = x^{\{0\}} P^n$$

Example 5.7 *A* 3×3 *Example:*

$$P_{33} = \begin{bmatrix} 0 & 0.4 & 0.6 \\ 0.5 & 0 & 0.5 \\ 0.25 & 0 & 0.75 \end{bmatrix}$$

```
P33 <- matrix(c(0, 0.4, 0.6,
              0.5, 0, 0.5,
              0.25, 0, 0.75), nrow = 3,
              byrow = TRUE)
```

```
P2 <- P33%*%P33
P4 <- P2%*%P2 # 4th power of P
P8 <- P4%*%P4 # 8th power of P
P16 <- P8%*%P8 # 16th power of P

x0 <- c(0, 1, 0)

x16 <- x0%*%P16 # distribution after 16 transitions
x16

##          [,1]       [,2]      [,3]
## [1,] 0.217389 0.08695851 0.6956525

x32 <- x16%*%P16 # distribution after 32 transitions
x32

##           [,1]       [,2]      [,3]
## [1,] 0.2173913 0.08695652 0.6956522
```

The distribution of X_n no longer seems to depend on n. We have found the limiting *stationary distribution* of this Markov chain.

Example - $x^{\{0\}}P_{33}$ and $x^{\{0\}}P_{33}^2$ with random starting point

```
x0 <- rep(1/3, 3)  # random starting point
x1 <- x0%*%P33  # distribution after 1 transition
x2 <- x0%*%(P33%*%P33) # distribution after 2 transitions
```

Example 5.8 *$x^{\{0\}}$, $x^{\{1\}}$ and $x^{\{2\}}$:*

```
x0

## [1] 0.3333333 0.3333333 0.3333333

x1

##      [,1]      [,2]      [,3]
## [1,] 0.25 0.1333333 0.6166667

x2

##           [,1] [,2]      [,3]
## [1,] 0.2208333  0.1 0.6791667
```

5.2.5 *Probability distribution of X_n when n is large*

5.2.6 *Does every Markov chain have a limiting stationary distribution?*

Not all Markov chains have limiting stationary distributions, but Markov chains with regular transition matrices do. A transition matrix P is said to be a regular if there exists some positive integer n such that all entries of P^n are greater than zero.

Example 5.9
```
P33%*%P33%*%P33 # 3rd power of P33

##          [,1]  [,2]     [,3]
## [1,] 0.162500 0.140 0.697500
## [2,] 0.268750 0.050 0.681250
## [3,] 0.228125 0.075 0.696875
```

P_{33} is a regular transition matrix.

Example 5.10 *P for the mouse odor example is not regular. Therefore, regularity is not always necessary for a stationary distribution to exist, but regularity is required for the Markov chain to converge to converge to the stationary distribution.*

5.2.7 Implications of regularity

If P is a regular matrix, then there exists a vector **q** such that

$$\lim_{n\to\infty} P^n = \begin{bmatrix} \mathbf{q} \\ \vdots \\ \mathbf{q} \end{bmatrix}$$

Example 5.11
```
P33power <- P33%*%P33
P33power <- P33power%*%P33power
P33power <- P33power%*%P33power
P33power <- P33power%*%P33power
P33power <- P33power%*%P33power
P33power

##           [,1]       [,2]      [,3]
## [1,] 0.2173913 0.08695652 0.6956522
## [2,] 0.2173913 0.08695652 0.6956522
## [3,] 0.2173913 0.08695652 0.6956522
```

$\mathbf{q} = [0.2173, 0.08695, 0.6956]$

Example 5.12
```
P16

##           [,1]       [,2]      [,3]
## [1,] 0.2173940 0.08695419 0.6956518
## [2,] 0.2173890 0.08695851 0.6956525
## [3,] 0.2173907 0.08695700 0.6956523

P16%*%P16%*%P16

##           [,1]       [,2]      [,3]
## [1,] 0.2173913 0.08695652 0.6956522
## [2,] 0.2173913 0.08695652 0.6956522
## [3,] 0.2173913 0.08695652 0.6956522
```

If P is a regular matrix, then there exists a unique vector **q** such that

$$\lim_{n\to\infty} x^{\{n\}} = \mathbf{q}$$

for any initial state vector $x^{\{0\}}$. The vector **q** specifies the limiting stationary distribution of the Markov chain.

If P is a regular matrix, then the limiting stationary distribution vector **q** is the unique solution to the equation

$$\mathbf{q} = \mathbf{q}P$$

whose entries sum to one. The solution of the above equation is the *stationary distribution* or stationary probability vector.

Example 5.13
```
q <- c(.2173913, .08695652, 0.6956522)
q%*%P33  # test to see if equality holds here

##            [,1]       [,2]      [,3]
## [1,] 0.2173913 0.08695652 0.6956522
```

Example 5.14 *This is a non-regular example, but*

```
q <- c(.1, .2, .4, .3)
q%*%P  # test to see if equality holds here

##      [,1] [,2] [,3] [,4]
## [1,]  0.1  0.2  0.4  0.3
```

the stationary distribution still satisfies

$$\mathbf{q} = \mathbf{q}P.$$

5.2.8 Law of large numbers for Markov chains

Theorem 5: If $X_1, X_2, \ldots$ is a finite state Markov chain with a regular transition matrix, then for any function $f(j)$ defined on the state space, and for any initial state X_0,

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} f(X_k) = \mathrm{q} \begin{bmatrix} f(1) \\ \vdots \\ f(m) \end{bmatrix} = \sum_{j=1}^{m} q_j f(j) \quad \text{with probability 1}$$

Note that

$$E[f(X)] = \sum_{j=1}^{m} q_j f(j)$$

when X has a distribution given by **q**. (One implication of this is Markov Chain Monte Carlo simulation, i.e. MCMC).

5.2.9 Law of Large Numbers for Periodic Markov Chains

The preceding result also holds for periodic Markov chains as long as the matrix of transitions within each periodic class is regular.

5.2.10 Law of Large Numbers for Periodic Markov Chains

Example: $f(x) = x^2$. Check result for Mouse Odor example:

```
mean(location^2) # location contains a simulated chain

## [1] 9.345

(1:4)^2%*%c(.1, .2, .4, .3)   #  expected value of f(X^2)

##       [,1]
## [1,]   9.3
```

The two results match.

5.2.11 *Classification of Markov Chain States*

The following discussion leads to a simple way of determining whether a transition matrix is regular or not.

Definition. State i *leads* to state j if there exists $n \geq 1$ such that $P_{ij}^{(n)} > 0$.

e.g. Mouse maze: State 1 leads to state 2; State 2 leads to state 1, etc.

Definition. States i and j are said to *communicate* if i leads to j and j leads to i.

e.g. States 1 and 2 communicate.

Proposition. The 'leads to' relation is transitive. That is, if i leads to j and j leads to k, then i leads to k.

Mouse Maze e.g.: State 1 leads to State 2, State 2 leads to State 3, so State 1 leads to State 3. Similarly, State 3 leads to State 1. Therefore, States 1 and 3 communicate. Similarly, State 1 and State 4 communicate.

Definition. A *class* of states is defined as a subset of S in which any two members communicate.

Mouse Maze e.g.: All states communicate. They form a class.

Example:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

States 1 and 2 communicate, but State 3 does not communicate with States 1 and 2. $\{1, 2\}$ is one class and $\{3\}$ is another class.

Definition. S is said to be *irreducible* if it is a class. That is, if all states in S communicate, S is said to be irreducible.

The mouse maze state space is irreducible, but the state space for P is not irreducible.

Definition. For any $i \in S$, the *period* of state i is defined to be the greatest common divisor of the set

$$\{n > 0 : P_{ii}^{(n)} > 0\}$$

Proposition. If i and j communicate, then the periods of i and j are the same.

Definition. If the state space of a Markov chain is irreducible, then the period of the Markov chain is defined to be the common period of each state.

Mouse Maze e.g.: Period is 2, since if the chain starts in State 1 it can never return to State 1 in an odd number of transitions, and all states communicate.

Definition. If the period of a Markov chain is 1, the Markov chain is said to be aperiodic.

P_{33} is aperiodic.

Theorem. An aperiodic irreducible Markov chain with a finite state space must have a regular transition matrix.

P_{33} is regular. P for the mouse maze example is not regular.

5.2.12 *Calculation of Steady State Vector*

The stationary vector $\mathbf{q}$ solves

$$\mathbf{q} = \mathbf{q}P$$

which can be re-written as

$$(P^\top - I)\pi = 0$$

where $\pi = \mathbf{q}^\top$.

```
P33 <- matrix(c(0, 0.4, 0.6,
0.5, 0, 0.5,
0.25, 0, 0.75), nrow = 3,
byrow = TRUE)
A <- t(P33) - diag(rep(1,3)) # P^T - I
solve(A, rep(0,3))    # solve A pi = 0

## Error in solve.default(A, rep(0, 3)):  Lapack routine dgesv:  system is exactly
singular:  U[3,3] = 0
```

trouble! We either have too many solutions or no solutions to this problem. $P^\top - I$ is singular, so there are too many solutions. We need more equations. Since the stationary probability vector is a (discrete) probability distribution, its elements must sum to 1:

$$\pi_1 + \pi_2 + \pi_3 = 1, \qquad \text{so we include this equation:}$$

```
A <- rbind(A, rep(1,3))
RHS <- c(rep(0,3), 1)
qr.solve(A, RHS)   # no longer a square system

## [1] 0.21739130 0.08695652 0.69565217
```

5.2.13 Solution of Linear Systems via QR (`qr.solve()`)

Every matrix A has a QR decomposition. Suppose A is $n \times m$ with $n > m$ (as in our case, where $n = m + 1$).

$$A = QR$$

where Q is an $n \times n$ orthogonal matrix, i.e. $Q^\top Q = I$, and R is an $n \times m$ upper triangular matrix. Among other things, this means that all entries of R's bottom $n - m$ rows are 0's. Now, solve

$$\begin{aligned} Ax &= y \\ \rightsquigarrow QRx &= y \\ \rightsquigarrow Q^\top QRx &= Q^\top y \\ \rightsquigarrow Rx &= Q^\top y \end{aligned}$$

which can be solved for x using backward substitution, starting at row m. If the last $n - m$ elements of $Q^\top y$ are 0, then the system has a solution. If not, the system has no exact solution, but the result is the least-squares estimate.

Again, we will code a function called `pstationary()` which we will call every time we wish to find a stationary probability distribution for a given transition matrix P.

```
pstationary <- function(P) {
    s <- ncol(P)
    A <- t(P) - diag(rep(1, s)) # P^T - I
    A <- rbind(A, rep(1, s))
    RHS <- c(rep(0, s), 1)
    qr.solve(A, RHS)
}
```

Example 5.15 *Recall the mouse-with-odor model. We can find the stationary probability distribution with our newly created function as follows.*

```
P <- matrix(c(0, 1, 0, 0,
                           0.5, 0, 0.5, 0,
                           0, 0.25, 0, 0.75,
                           0, 0, 1, 0),
nrow = 4,  byrow = TRUE)  # P is the transition matrix
pstationary(P)

## [1] 0.1 0.2 0.4 0.3
```

Example 5.16 Symmetric random walk with reflecting barrier.
Suppose $S = \{0, \pm1, \pm2, \ldots, \pm k\}$ for some $k > 2$.

$$X_n = X_{n-1} + 2B_n - 1 \tag{5.2}$$

where B_n is Bernoulli with parameter $p = .5$, independent of X_{n-1}. When $X_{n-1} = \pm k$, $X_n = k - 1$ (or $1 - k$).
We will compute the stationary distribution for the case when $k = 3$. For this case, the transition matrix P is 7×7.

```
k <- 3;  s <- 2*k + 1
P <- matrix(c(
    0.0,1.0,0.0,0.0,0.0,0.0,0.0,
    0.5,0.0,0.5,0.0,0.0,0.0,0.0,
    0.0,0.5,0.0,0.5,0.0,0.0,0.0,
    0.0,0.0,0.5,0.0,0.5,0.0,0.0,
    0.0,0.0,0.0,0.5,0.0,0.5,0.0,
    0.0,0.0,0.0,0.0,0.5,0.0,0.5,
    0.0,0.0,0.0,0.0,0.0,1.0,0.0
), nrow = s, byrow = TRUE)
```

```
pi <- pstationary(P)
names(pi) <- seq(-3, 3)
```

The stationary distribution is

```
pi

##     -3     -2     -1      0      1      2      3
## 0.0833 0.1667 0.1667 0.1667 0.1667 0.1667 0.0833
```

Simulating 100000 steps of the random walk can then be accomplished using our `rMC` *function.*

```
N <- 100000
X <- rMC(N = N, initial.state = 1, P = P)
pi.sim <- table(X)/N
names(pi.sim) <- seq(-3, 3)
pi.sim

##     -3     -2     -1      0      1      2      3
## 0.0842 0.1697 0.1691 0.1654 0.1650 0.1649 0.0817
```

The simulated relative frequencies match the theoretical stationary probabilities to two decimal places. A larger number of transitions will increase the accuracy.

When transition matrices have a repeating structure, it can be inefficient to code up the entire matrix. For the random walk, it is somewhat more efficient to exploit the structure described by equation (5.2) and to simulate the transitions or moves that the random walk goes through at each successive time step. The size of the move at the nth step is $2B_n - 1$ which is efficiently simulated in R using the `sample()` function applied to the set $\{-1, 1\}$. When the random walk X_n is at the boundary (either 0 or 6), it moves back to either 1 or 5. Thus, $X_{n+1} = |X_n - 1|$. Simulating the random walk for 100000 transitions proceeds as follows:

```
N <- 100000
X <- numeric(N)#initializing
current.state <- 1 # initial state
for (n in 1:N) {
    if (current.state == 0 | current.state == 6) {
       current.state <- abs(current.state - 1)
    } else {
    move <- sample(c(-1, 1), size = 1)
    current.state <- current.state + move
    }
    X[n] <- current.state
}
```

We next compute the relative frequency distribution of the numbers of visits to the 7 states using the `table()` function so that we can compare with the stationary distribution predicted by the theory.

```
pi <- table(X)/N;  names(pi) <- seq(-3, 3); pi

##     -3     -2     -1      0      1      2      3
## 0.0837 0.1672 0.1674 0.1678 0.1669 0.1650 0.0819
```

Again, we have achieved two digit accuracy. This relatively slow convergence is typical of Markov chain simulation methods, since the simulated observations are, by design, dependent on each other. This dependence reduces the effective amount of information in the data. If we simulate directly 100000 independent observation from the theoretical distribution, we should expect three digit accuracy in the probability estimates. However, direct simulation of the stationary distribution cannot give us information such as the distribution of run lengths, whereas the simulation of the Markov chain provides such information. Figure 5.4 displays a relative frequency histogram of numbers of transitions between visits to State 5 for the Markov chain simulated in Example 5.16. The Run Length Encoding function `rle()` is used to count run lengths for the sequence in `X`. Unsurprisingly, the distribution is concentrated on the runlengths which are between 0 and 10, but runs in excess of 50 occur with a larger probability than might have been expected.

```
Xnot5 <- rle(X != 5)
Xnot5 <- Xnot5$lengths[Xnot5$values]
hist(Xnot5, freq = FALSE, xlab = "Run Lengths", main = " ")
```

The book by Tijms (1994) contains many examples showing how Markov chains arise in practical situations. The following example is a slight modification of one of these.

Example 5.17 *At the beginning of each day, a batch of containers arrives at a stockyard having capacity to store 6 containers. The batch size has the discrete probability distribution $\{q_0 = .4, q_1 = 0.3, q_2 = 0.2, q_3 = 0.1\}$. If the stockyard does not have sufficient space to store the whole batch, the batch as a whole is taken elsewhere. Each day, as long as there are containers in the stockyard, exactly one container is removed from the stockyard.*

We will address the following questions:

1. *Find the transition matrix for the Markov chain $\{X_1, X_2, \ldots\}$, where X_t = the number of containers in the stockyard at the beginning of the tth day.*

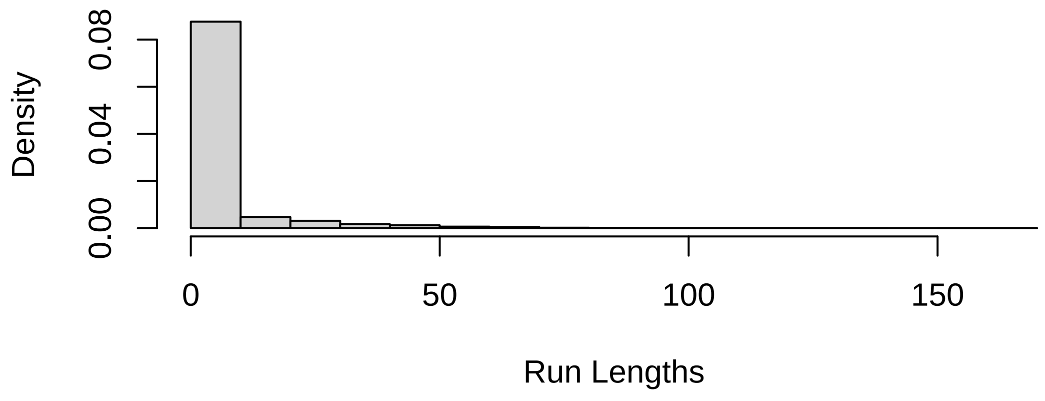


Figure 5.4: Empirical distribution of runs avoiding State 5 in N transitions of a 7-state random walk.

2. *Find the limiting stationary distribution for this Markov chain.*
3. *Suppose a profit of $100 is realized for each container that spends a night at the stockyard. Calculate the long-run average daily profit.*

We first find the transition matrix P.
Let X_t = the number of containers at the start of day t. The state space is

$$S = \{0, 1, 2, 3, 4, 5\}.$$

Note that the stockyard could be empty at the beginning of a day, and because it can only hold 6 containers, it could never end a day with more than 5 containers, since 1 is always taken away. Careful consideration then leads to

$$P = \begin{bmatrix} 0.7 & 0.2 & 0.1 & 0 & 0 & 0 \\ 0.4 & 0.3 & 0.2 & 0.1 & 0 & 0 \\ 0 & 0.4 & 0.3 & 0.2 & 0.1 & 0 \\ 0 & 0 & 0.4 & 0.3 & 0.2 & 0.1 \\ 0 & 0 & 0 & 0.5 & 0.3 & 0.2 \\ 0 & 0 & 0 & 0 & 0.7 & 0.3 \end{bmatrix}$$

The steady-state distribution can be found with the code

```
P <- matrix(c(0.7, .2, .1, 0, 0, 0, 0.4, 0.3, 0.2, 0.1,
0, 0, 0, 0.4, 0.3, 0.2, 0.1, 0, 0, 0, 0.4, 0.3, 0.2, 0.1,
0, 0, 0, 0.5, 0.3, 0.2,  0, 0, 0, 0, 0.7, 0.3),
nrow=6, byrow = TRUE)
A <- t(P) - diag(rep(1, 6)) # P^T - I
A <- rbind(A, rep(1,6))
RHS <- c(rep(0,6), 1)
options(digits=4)
pi <- qr.solve(A, RHS)
pi

## [1] 0.23273 0.17455 0.18909 0.18545 0.14909 0.06909
```

Finally, the expected long-run daily profit is found using the steady-state distribution and by defining

$$\text{Profit } = 100 \times X$$

where X is the number of containers in the yard at the beginning of a day. Hence,

$$E[X] = \sum_{i=0}^{5} i\pi_i$$

and we have

```
sum(pi*(0:5))

## [1] 2.051
```

so

$$E[100X] = 205.10$$

Exercises

1. The daily weather in a certain area is either sunny (S) or cloudy. Rain (R) occurs on half of the cloudy days. There is no precipitation on the other cloudy days (C). If it is sunny on a given day, it will be sunny the next day with probability 0.5. Otherwise, it will become cloudy. The day after a rainy cloudy day is always cloudy, but the day after a dry cloudy day is sunny with probability 0.4.

   (a) Suppose today is sunny. What is the probability that tomorrow will be cloudy and dry? What is the probability that tomorrow will be cloudy and rainy?

   (b) Suppose today is cloudy and rainy? What is the probability that tomorrow will be cloudy and rainy? What is the probability that tomorrow will be cloudy and dry?

   (c) Suppose today is cloudy and dry? What is the probability that tomorrow will be cloudy and rainy? What is the probability that tomorrow will be cloudy and dry?

   (d) Supposing that the weather in the area can be modelled as a Markov chain, write out the transition matrix, identifying which rows correspond to which of the 3 states.

   (e) Hence, or otherwise, find the probability that the day after tomorrow will be sunny, given that today is sunny.

2. Consider the Markov chain $X_1, X_2, \ldots,$ with transition matrix

$$\mathbf{P} = \begin{bmatrix} 0 & 0.1 & 0.9 \\ 0.5 & 0 & 0.5 \\ 0.8 & 0.2 & 0 \end{bmatrix}.$$

   (a) Suppose $X_1 = 3$. Simulate the values of X_2, X_3, X_4, X_5, and X_6 using `set.seed(92341)` and the function

```
MC.sim <- function(n,P,x1) {
    sim <- as.numeric(n)
    m <- ncol(P)
    if (missing(x1)) {
        sim[1] <- sample(1:m,1)
    } else {
        sim[1] <- x1
    }
    for (i in 2:n) {
        newstate <- sample(1:m,1,prob=P[sim[i-1],])
        sim[i] <- newstate
    }
sim
}
```

(b) Does state 1 lead to state 3?

(c) Does state 3 lead to state 1?

(d) Do states 1 and 3 communicate?

(e) Is the state space irreducible?

(f) If there is a stationary distribution, find it.

3. Consider the Markov chain $X_1, X_2, \ldots,$ with transition matrix

$$\mathbf{P} = \begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \end{bmatrix}.$$

(a) Suppose $X_1 = 1$. Simulate the values of X_2, X_3, X_4, X_5, X_6 and X_7 using `set.seed(135)`.

(b) Does state 1 lead to state 3?

(c) Does state 3 lead to state 1?

(d) Do states 1 and 3 communicate?

(e) Is the state space irreducible?

(f) If there is a stationary distribution, find it.

4. A Markov chain $\{X_n, n \geq 0\}$ is defined on the state space $\{1, 2, 3\}$ with transition matrix P.

$$P = \begin{bmatrix} .3 & .6 & .1 \\ .75 & 0 & .25 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) Which states lead to state 3?

(b) Which state(s) is/are absorbing?

(c) Which states communicate?

5. A Markov chain $\{X_n, n \geq 0\}$ is defined on the state space $\{1, 2, 3, 4\}$ with transition matrix P.

$$P = \begin{bmatrix} .9 & .1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(a) Which states lead to state 1?

(b) Which state(s) is/are absorbing?

(c) Which states communicate?

(d) Find the stationary distribution if it exists.

6. A Markov chain $\{X_n, n \geq 0\}$ is defined on the state space $\{1, 2, 3, 4\}$ with transition matrix P. Suppose

$$P = \begin{bmatrix} .2 & .1 & .3 & .4 \\ .2 & 0 & .8 & 0 \\ .6 & .4 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

   (a) Which states communicate?
   (b) Find the stationary distribution if it exists.

7. The following is a transition matrix for another Markov chain $X_1, X_2, \ldots$ on the state space {1, 2, 3, 4, 5, 6}:

```
##       [,1] [,2]  [,3] [,4]  [,5]  [,6]
## [1,] 0.750 0.25 0.000 0.00 0.000 0.000
## [2,] 0.125 0.75 0.125 0.00 0.000 0.000
## [3,] 0.000 0.25 0.500 0.25 0.000 0.000
## [4,] 0.000 0.00 0.250 0.50 0.250 0.000
## [5,] 0.000 0.00 0.000 0.25 0.500 0.250
## [6,] 0.000 0.00 0.000 0.00 0.125 0.875
```

   (a) Verify that the stationary distribution for this Markov chain is

$$\pi = [1/8\ 1/4\ 1/8\ 1/8\ 1/8\ 1/4].$$

   (b) Does state 1 communicate with state 6? Explain clearly.
   (c) If the Markov chain was simulated for a very long time, what would you expect the long-run average of the X's ($\frac{1}{n}\sum_{i=1}^{n} X_i$) to be?

8. A dam with a volume of 10 units is to be constructed on a river. Water flows into the dam each day is according to a probability distribution given by $p_0 = .3, p_1 = .1, p_2 = .3, p_3 = .2$, and $p_4 = .1$. Any excess water over the 10 unit capacity drains away and is lost. Water requirements are 2 units in each day, and the water must be available at the beginning of the day. On any day when an insufficient amount of water is available, the shortfall is made up at a cost of 5 per unit. The cost of operating the dam is the equivalent of 2 units.

   (a) Find the transition probability matrix, P, for the Markov chain $\{X_0, X_1, X_2, \ldots\}$, where $X_t =$ the volume of water available at the beginning of the tth day.
   (b) Is the transition matrix regular?
   (c) What is the long-run steady-state distribution for this Markov chain?
   (d) Find the long-run average daily cost of supplying water.
   (e) Starting from the assumption that $X_1 = 1$, write code which simulates this Markov chain for 100000 days.
   (f) Use your simulation result to estimate the probability that the annual cost of supplying water could exceed 1500 units.

9. Recall the single-player Monopoly simulation from the beginning of the book. This problem concerns a greatly simplified version of the game, where a player will move a number of spaces around a 40 space rectangular board, corresponding to what appears on a pair of rolled dice. Thus, there are 40 states in the state space, with the first space corresponding to "Go". The 11th space corresponds to "Jail" and the 31st space corresponds to "Go to Jail".

   (a) Construct a 40×40 transition matrix for this Markov chain, based on the dice rolls as well as using the fact that when the player hits space 31, the probability is 1 that the player moves to space 11 at the next move. (This rule is slightly different from the usual rules, but makes analysis slightly easier with little loss of accuracy. We also assume that the player leaves space 11 at the next move, unlike the actual rules of the game.)
   (b) Find the limiting stationary distribution for this Markov chain, calling the resulting state vector `PI`. Use

```
names(PI) <- 1:40
```

to attach names to the vector, and display the probabilities with a bar plot. Which space is second-most frequently visited? How often is space 40 visited?

(c) Suppose you have built hotels on spaces 17, 19 and 20, and you can receive revenue of \$950, \$950 and \$1000 for each time that your opponent lands on these spaces. Calculate the long run average amount of revenue (per turn) that you could obtain from your opponent?

(d) Suppose your opponent has built hotels on spaces 38 and 40, and it will cost you \$1500 and \$2000 each time you land on one of these spaces. Calculate the long run average cost per turn. By subtraction, you can find the expected profit per turn for someone who owns hotels on spaces 17, 19 and 20 while playing against someone with hotels on spaces 38 and 40.

(e) Calculate the long run variance of your costs and of your revenues. Add these variances together to obtain the variance of your profit. Finally, take the square root to obtain the standard deviation of your profit. Interpret your results.

(f) Repeat the previous three exercises, under the assumption that you additionally have built hotels on spaces 22, 24 and 25 and can generate revenue \$1050, \$1050, and \$1100 at these locations, and your opponent has additionally built hotels on spaces 32, 33 and 35 which will cost you \$1275, \$1275 and \$1400 when you land on those spaces.

(g) Under the assumptions of (c) and (d), write code that will simulate concurrent Markov chains for you and your opponent, both starting in space 1, with both starting with a cash surplus of \$5000, and ending when the first person loses all of their cash. Run this simulation 1000 times, and estimate the probability that you would win. Re-run the simulation 1000 times under the assumptions in (f), and estimate the probability that you would win.

10. Refer to the bivariate Bernoulli random variables X_1 and X_2 defined in Exercise 1 of Section 4.1. Define a sequence of Bernoulli random variables $Y_1, Y_2, \ldots$ for which $P(Y_{2n} = 0|Y_{2n-1} = 0) = P(X_2 = 0|X_1 = 0)$ and $P(Y_{2n} = 0|Y_{2n-1} = 1) = P(X_2 = 0|X_1 = 1)$, and $P(Y_{2n+1} = 0|Y_{2n} = 0) = P(X_1 = 0|X_2 = 0)$ and $P(Y_{2n+1} = 0|Y_{2n} = 1) = P(X_1 = 0|X_2 = 1)$, for $n = 1, 2, \ldots$. Let $Z_n = Y_{2n}$ for $n = 1, 2, \ldots$, and suppose Z_0 is a given Bernoulli random variable.

(a) Argue that the sequence $\{Z_n\}$ is a Markov chain and show that the transition matrix is

$$P = \begin{bmatrix} \frac{p_1^2}{(p_1+p_3)(p_1+p_2)} + \frac{p_3^2}{(p_1+p_3)(p_4+p_3)} & \frac{p_1 p_2}{(p_1+p_3)(p_1+p_2)} + \frac{p_3 p_4}{(p_1+p_3)(p_4+p_3)} \\ \frac{p_4 p_3}{(p_4+p_3)(p_4+p_2)} + \frac{p_2 p_1}{(p_1+p_2)(p_4+p_2)} & \frac{p_4^2}{(p_4+p_3)(p_4+p_2)} + \frac{p_2^2}{(p_1+p_2)(p_4+p_2)} \end{bmatrix},$$

(b) Show that the stationary distribution for this Markov chain is $[p_1 + p_3 \quad p_2 + p_4]$.

(c) Deduce that when Gibbs sampling is applied to the bivariate Bernoulli distribution as described in Section 4.1, the sequence of X_2 values generated follow a distribution that is well approximated by the true marginal distribution of X_2. Conclude that since the true conditional distribution of X_1, given X_2 is used to generate the X_1 values, the simulated pairs of X_1 and X_2 values have a distribution that well approximates the true joint distribution of X_1 and X_2.

5.3 Markov chain Monte Carlo

Markov Chain Monte Carlo simulation methods, or MCMC as they are popularly referred to, are a powerful set of tools. MCMC is widely recognized as having changed the status of Bayesian data analysis from a specialized area of statistical research to a major pillar of data science, because it provides computationally intensive solutions to problems which were earlier seen as completely intractable.

The book by Gelman et al. (2004) provides a comprehensive introduction to Bayesian statistics, including a detailed chapter on MCMC. The much shorter treatment given in one of the chapters of the book by Wood (2015) provides a quick path into the subject, highlighting many of the pitfalls that may arise when employing MCMC.

The goal of the present section is to explore the particular MCMC simulator, called the Metropolis-Hastings algorithm, in the context of Markov chain models and to provide some simple illustrative examples of its use. A similar, but perhaps more terse, treatment of the theory is provided in the book by Ross (2014).

5.3.1 Reversible Markov chains

In order to understand the Metropolis-Hastings algorithm, the concept of time-reversibility of a Markov chain is useful. This concept is also referred to as detailed balance.

A Markov chain with transition matrix P and stationary distribution π, such that $P(X_n = i) = \pi_i$ for all $n \in (-\infty, \infty)$, is said to be *time-reversible* if the Markov property holds for the chain when it is time reversed:

$$P(X_n = x_n | X_{n+1} = x_{n+1}, \ldots) = P(X_n = x_n | X_{n+1} = x_{n+1})$$

and the joint probabilities of the form $P(X_{n+1} = j, X_n = i)$ are the same as $P(X_n = j, X_{n+1} = i)$. It then follows that a Markov chain with transition matrix P is reversible if there exists a vector **q** such that

$$q_j P_{ji} = q_i P_{ij}.$$

For such a **q**, observe what happens when we multiply **q** by P. The ith component of the vector $\mathbf{q}P$ is

$$\sum_{j=1}^{\infty} q_j P_{ji}$$

and this is

$$q_i \sum_{j=1}^{\infty} P_{ij} = q_i$$

if the Markov chain is reversible. Therefore

$$\mathbf{q}P = \mathbf{q}.$$

which tells us that **q** is the stationary distribution for P. That is, $\mathbf{q} = \pi$.

Example 5.18 *We will now show that the symmetric random walk with reflecting barrier considered in Example 5.16 is time-reversible.*

For $|j| < k$,

$$P_{j,j+1} = P_{j,j-1} = 0.5.$$

and

$$P_{k,k-1} = 1 = P_{-k,-k+1}.$$

$$q_k P_{k,k-1} = q_{k-1} P_{k-1,k} \quad \textit{or}$$

$$q_k = q_{k-1} \times 0.5$$

and similarly,

$$q_{-k} = q_{1-k} \times 0.5.$$

For $|j| < k-1$,

$$q_j P_{j,j+1} = q_{j+1} P_{j+1,j} \quad \textit{or}$$

$$0.5 q_j = 0.5 q_{j+1}.$$

Therefore, all q's other than the q_k and q_{-k} are equal, and have twice the value of q_k and q_{-k}. This Markov chain is time-reversible.

Once we have shown that the Markov chain is time-reversible, we can use the associated **q** vector to find the steady-state distribution.

Example 5.19 *For the previous example, the steady-state distribution is given by*

$$q_j = \frac{1}{2(k-1)+2} = \frac{1}{2k}$$

and

$$q_k = q_{-k} = \frac{1}{4k}$$

For the special case where $k = 3$, we have

$$q_3 = q_{-3} = \frac{1}{12}$$

$$q_2 = q_1 = q_0 = q_{-1} = q_{-2} = \frac{1}{6}.$$

This result agrees with what was obtained from solving the linear system given by $\pi P = \pi$.

5.3.2 Other time-reversible Markov chains

Suppose $\{\pi_i, i = 0, \pm 1, \pm 2, \ldots\}$ is a set of positive real numbers with $\sum_{i=-\infty}^{\infty} \pi_i = 1$. (This is a probability distribution on the integers.)

Set

$$P_{i,j} = \frac{1}{6} \min\left(\frac{\pi_j}{\pi_i}, 1\right), \text{ for } j = i-2, i-1, i+1, i+2$$

and 0 for $|j - i| > 2$. $P_{i,i}$ is taken to be the value that ensures that the ith row sum of P is 1.

To verify that the Markov chain is reversible, show that

$$\pi_i P_{i,i+2} = \pi_{i+2} P_{i+2,i}$$

and so on. This is an example of an infinite-state time-reversible Markov chain. Note that the steady-state vector has infinite length and has ith entry π_i.

We can simulate from this infinite state Markov chain by simply keeping track of the current state after each transition and updating the probability distribution for the next transition accordingly.

Example 5.20 *Suppose $\pi_i = k/(i+1)^4$ for $i > 0$ and $\pi_i = 0$ for all $i < 1$. The proportionality constant k ensures that $\sum_{i=1}^{\infty} \pi_i = 1$. Note that we can simulate from this Markov chain without knowing k.*

```
pi.fun <- function(i) {
    out <- 0
    if (i > 0) out <- 1/(i+1)^4
    out
}
```

```
N <- 20000
X <- numeric(N)
current.state <- 50  # initialize the Markov chain
for (n in 1:N) {
     i <- current.state
     P <- c(min(pi.fun(i-2)/pi.fun(i), 1),
     min(pi.fun(i-1)/pi.fun(i), 1),
     min(pi.fun(i+1)/pi.fun(i), 1),
     min(pi.fun(i+2)/pi.fun(i), 1))/6
     P0 <- 1 - sum(P)
     P <- c(P[1:2], P0, P[3:4])
     transition <- sample(seq(-2,2,1), size = 1, prob = P)
     current.state <- current.state + transition
     X[n] <- current.state
}
```

The relative frequency distribution can be computed using the `table()` *function after removing a sufficient number of the early observations.*

```
n <- 1000
observedDist <- table(X[-c(1:n)])
observedDist/(N-n)

##
##            1            2            3            4            5
## 7.324737e-01 1.286316e-01 4.694737e-02 1.963158e-02 1.473684e-02
##            6            7            8            9           10
## 7.263158e-03 5.736842e-03 3.578947e-03 2.736842e-03 2.052632e-03
##           11           12           13           14           15
## 1.000000e-03 8.947368e-04 4.210526e-04 4.736842e-04 2.631579e-04
##           16           17           18           19           20
## 3.157895e-04 2.105263e-04 6.315789e-04 5.263158e-05 5.789474e-04
##           21           22           23           24           25
## 2.105263e-04 4.736842e-04 3.684211e-04 5.789474e-04 2.631579e-04
##           26           27           28           29           30
## 3.157895e-04 5.263158e-04 6.842105e-04 5.789474e-04 6.842105e-04
##           31           32           33           34           35
## 1.210526e-03 8.421053e-04 8.947368e-04 1.000000e-03 7.368421e-04
##           36           37           38           39           40
## 8.947368e-04 6.842105e-04 4.210526e-04 8.947368e-04 1.210526e-03
##           41           42           43           44           45
## 2.000000e-03 8.421053e-04 1.000000e-03 5.263158e-04 8.421053e-04
##           46           47           48           49           50
## 7.368421e-04 6.842105e-04 2.105263e-04 5.789474e-04 5.789474e-04
##           51           52           53           54           55
## 5.789474e-04 3.157895e-04 5.789474e-04 5.263158e-04 1.578947e-04
##           56           57           58           59           60
## 1.052632e-04 2.105263e-04 3.157895e-04 3.684211e-04 4.736842e-04
##           61           62           64           65           66
## 4.210526e-04 2.631579e-04 2.105263e-04 2.105263e-04 4.210526e-04
##           67           68           69           70           71
## 5.263158e-05 2.105263e-04 1.052632e-04 3.684211e-04 4.210526e-04
##           72           73           74           75           76
## 4.736842e-04 3.684211e-04 3.684211e-04 3.684211e-04 1.052632e-04
##           77           78           79           80           81
## 6.315789e-04 5.789474e-04 4.736842e-04 1.578947e-04 5.263158e-05
```

This distribution serves as an estimate of the distribution π.

5.3.3 Burn-In

In the example, we omitted the first 1000 observations. This was to allow the Markov chain to achieve steady state, i.e., for it to be well approximated by the limiting stationary distribution.

```
ts.plot(X)
```

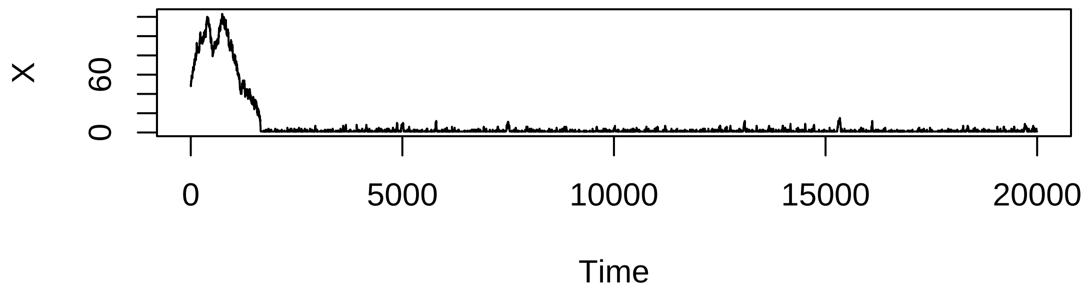


Figure 5.5: Trace plot of all simulated values from a Markov chain including the initial burn-in period.

We can see Figure 5.5 that the Markov chain starts off far from the stationary distribution which has probability mass concentrated at values mostly below 10. After about 500 transitions, this trace seems to show that the Markov chain is “mixing” well.

5.3.4 Estimating the proportionality constant k

Note that

$$\pi_2 = k/(2+1)^4 = k/81$$

so an estimate of k can be obtained by multiplying the observed probability of a 2 by 81:

```
k <- observedDist[2]/19000*81
k

##        2
## 10.41916
```

5.3.5 Markov chain Monte Carlo simulation via Metropolis-Hastings

This procedure is one version of MCMC, developed by Metropolis and Hastings. The procedure is as follows.

1. Given a distribution π, known up to a proportionality constant (k), find a time-reversible Markov chain with π as the stationary distribution vector.

2. Simulate from that Markov chain.

3. After simulating for a long enough period (burn-in), the observed values of the Markov chain follow the limiting stationary distribution, i.e. π.

A function which carries out random walk Metropolis-Hastings for a discrete probability distribution π which is evaluated from a function called `pi.fun()` is given in the function `rMCMC()` below. This function simulates n steps of a Markov chain whose limiting stationary distribution is proportional to π beginning with an initial state which can be randomly generated from the actual stationary distribution, when possible, or chosen arbitrarily. In the former situation, no burn-in period would be necessary, since the chain is stationary from the start. The function also includes a parameter `span` which controls the number of nonzero values in each row of the transition matrix. For example, `span = 3` corresponds to a matrix whose jth row has 3 nonzero entries on each side of the diagonal element, for $j > 3$. Additional parameters controlling the output from `pi.fun()` can also be specified in the call to `rMCMC()`.

```
rMCMC <- function(n, initial.state, span = 2, ...) {
    current.state <- initial.state
    X <- numeric(n)
for (j in 1:n) {
     i <- current.state
     ispan <- (i-span):(i+span)
     pispan <- pi.fun(ispan, ...)
     middleindex <- span+1
     P <-  pmin(pispan/pispan[middleindex], 1)
     P[middleindex] <- 0
     P <- P/ceiling(sum(P))
     P[middleindex] <- 1 - sum(P)
     transition <- sample(seq(-span,span,1), size = 1, prob = P)
     current.state <- current.state + transition
     X[j] <- current.state
}
X
}
```

Example 5.21 Simulating from the stationary distribution of a queue having s servers

More information on these and related models can be found in the text of Hillier and Lieberman (2021).

```
pi.fun <- function(x, s = 10, rho = 9) {
    prob <- numeric(length(x))
    xles <- (x <= s) & (x >= 0)
    xmor <- (x > s)
    xless <- x[xles]; xmore <- x[xmor]
    prob[xles] <- (rho)^xless/factorial(xless)
    prob[xmor] <- (rho/s)^xmore*s^s/factorial(s)
    return(prob)
}
```

A time comparison with crude simulation shows that for large-scale problems, MCMC is usually the faster method.

Example 5.22 *For the simulation comparison, we*

```
maxQ <- 10000
probs <- numeric(maxQ)
probs <- pi.fun(0:maxQ)
cprobs <- cumsum(probs)
cprobs <- cprobs/max(cprobs)

pQ <- function(x)  {
    return(cprobs[x+1])
}

rQ <- function(n) {
    U <- runif(n)
    X <- numeric(n)
    x <- 0
    for (x in 0:(maxQ-1)) {
        X[U >= pQ(x)] <- x + 1
        x <- x+1
    }
    return(X)
}
```

```
N <- 10000
system.time(x <- rQ(N))

##    user  system elapsed
##   0.294   0.008   0.302
```

```
system.time(X <- rMCMC(N, rQ(1), span=50, rho=9))

##    user  system elapsed
##   0.563   0.000   0.563
```

Since we `rQ()` *simulates from an excellent approximation to the stationary distribution of the queue length, we can initialize the MCMC simulator, by calling* `rQ()` *once. By simulating the first Markov chain observation directly from the stationary distribution, there is no burn-in period; the Markov chain starts off in steady state.*

As can be seen, the MCMC run simulated the same number of random variates from the stationary distribution as the crude simulator, but in less than 20% of the time. The resulting random variates follow the same distribution as can be seen in the QQ-plot displayed in Figure 5.6.

```
qqplot(x, X); abline(0,1)
```

The Law of Large Numbers for regular Markov chains allows us to estimate quantities such as $E[X]$ and $E[g(X)]$ for given functions $g(x)$ by calculating

$$\frac{1}{N}\sum_{n=1}^{N} X_n \text{ and } \frac{1}{N}\sum_{n=1}^{N} g(X_n).$$

Example 5.23 *we need a couple examples here to demonstrate this result.*

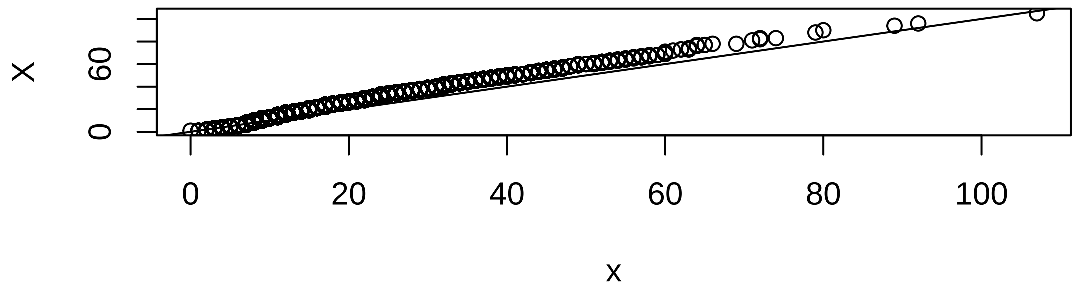


Figure 5.6: QQ-plot of simulated queue length variates coming from crude simulation compared with those coming from MCMC.

5.3.6 *MCMC Application - Bayesian Statistics*

We illustrate the practical application with a toy example first, followed by a slightly more realistic example.

Example 5.24 *Suppose N is Poisson distributed with mean 20, and given N, X is binomially distributed with parameters N and $p = 0.5$. N is not observed, but suppose $X = 5$.*[1] *Use MCMC to simulate the distribution of N, given X.*

Some terminology: the Poisson distribution for N is the prior distribution. The distribution of N, given $X = 5$, is called the posterior distribution. We create a function which computes the unnormalized posterior distribution for this problem.

```
posterior.fun <- function(i, x) {
    out <- 0
    if (i >= x) out <- dpois(i, lambda = 20)*
             dbinom(x, size = i, prob = .5)
    out
}
```

The following function calls the posterior distribution function in a way that is consistent with our earlier use of the Metropolis-Hastings procedure.

```
pi.fun <- function(i) {
    posterior.fun(i, x=5)
}
```

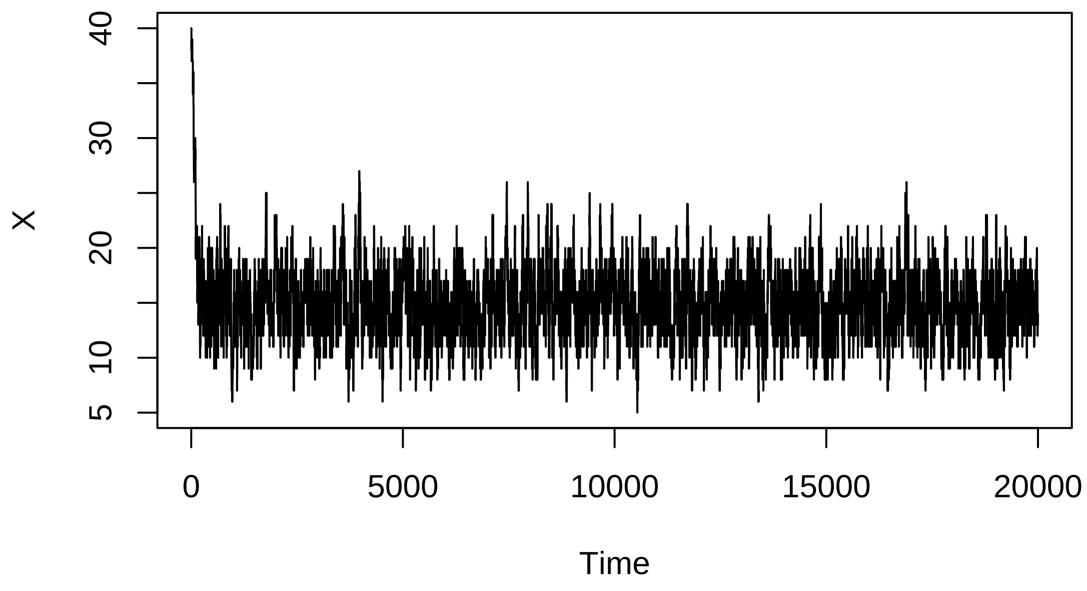
We now simulate the Markov chain as before.

```
N <- 20000
X <- numeric(N)
current.state <- 40  # initialize the Markov chain
for (n in 1:N) {
     i <- current.state
```

[1]This problem is somewhat artificial, but it can be viewed as arising from a coin-flipping experiment where the number of heads, X, has been recorded, but the number of coin flips was forgotten or not recorded in any case.

```
    P <- c(min(pi.fun(i-2)/pi.fun(i), 1),
           min(pi.fun(i-1)/pi.fun(i), 1),
           min(pi.fun(i+1)/pi.fun(i), 1),
           min(pi.fun(i+2)/pi.fun(i), 1))/6
    P0 <- 1 - sum(P)
    P <- c(P[1:2], P0, P[3:4])
    transition <- sample(seq(-2,2,1), size = 1, prob = P)
    current.state <- current.state + transition
    X[n] <- current.state
}
observedDist <- table(X[-c(1:1000)])
```

```
ts.plot(X)
```



The posterior distribution of N is then given by

```
options(width=50)
observedDist

##
##    5    6    7    8    9   10   11   12   13   14
##    2   15   82  242  446  754 1222 1663 2195 2463
##   15   16   17   18   19   20   21   22   23   24
## 2445 2158 1841 1415  884  508  261  190  114   49
##   25   26   27
##   39   11    1
```

A better way to observe the posterior distribution is through a bar plot such as in Figure 5.7.

```
par(mfrow=c(1,2))
theoryDist <- 20000*dpois(0:30, lambda = 20)
names(theoryDist) <- 0:30
barplot(theoryDist, main = "Prior")
barplot(observedDist, main = "Posterior")
```

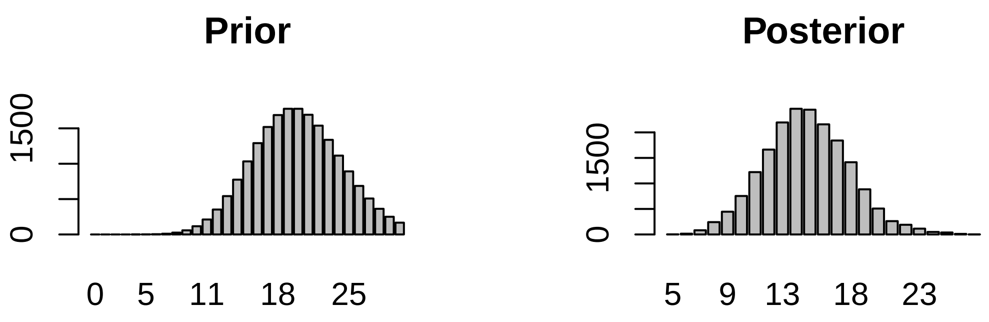


Figure 5.7: Prior and posterior distributions of N, before and after accounting for the observed data.

This is how the data $X = 5$ influences our belief (initially, Poisson(20)) about the distribution of the unknown value N.

5.3.7 What if our Prior Belief was Different?

e.g. $\lambda = 4$:

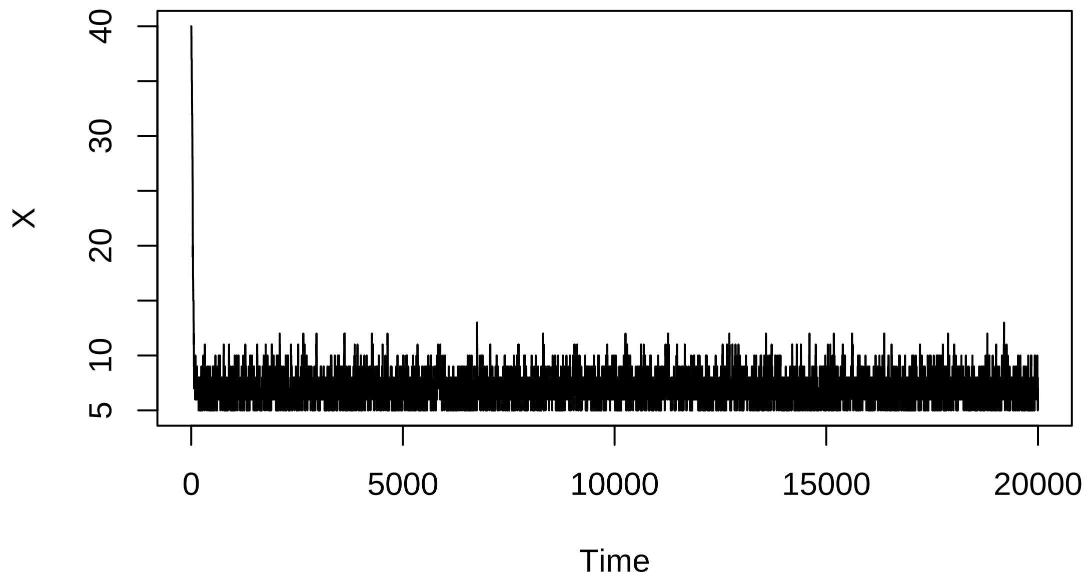
```
posterior.fun <- function(i, x) {
    out <- 0
    if (i >= x) out <- dpois(i, lambda = 4)*
             dbinom(x, size = i, prob = .5)
    out
}
```

5.3.8 Simulating the Markov Chain

```
N <- 20000
X <- numeric(N)
current.state <- 40  # initialize the Markov chain
for (n in 1:N) {
     i <- current.state
     P <- c(min(pi.fun(i-2)/pi.fun(i), 1),
               min(pi.fun(i-1)/pi.fun(i), 1),
               min(pi.fun(i+1)/pi.fun(i), 1),
               min(pi.fun(i+2)/pi.fun(i), 1))/6
     P0 <- 1 - sum(P)
     P <- c(P[1:2], P0, P[3:4])
     transition <- sample(seq(-2,2,1), size = 1, prob = P)
     current.state <- current.state + transition
     X[n] <- current.state
}
observedDist <- table(X[-c(1:1000)])
```

5.3.9 Plotting the Trace

```
ts.plot(X)
```



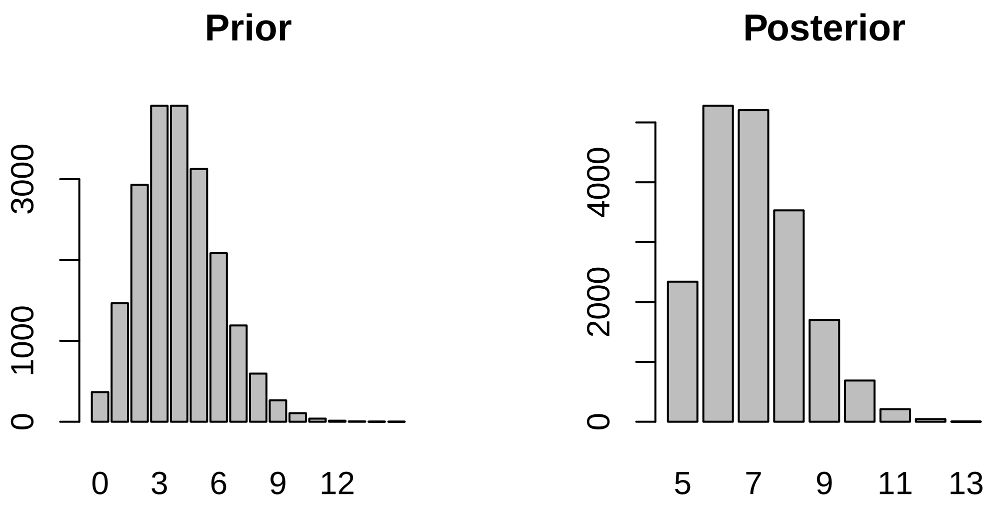
5.3.10 Posterior Distribution of N

```
options(width=50)
observedDist

##
##    5    6    7    8    9   10   11   12   13
## 2340 5278 5205 3531 1701  688  210   44    3
```

```
par(mfrow=c(1,2))
theoryDist <- 20000*dpois(0:15, lambda = 4)
```

```r
names(theoryDist) <- 0:15
barplot(theoryDist, main = "Prior")
barplot(observedDist, main = "Posterior")
```



This is how the *data* $X = 5$ influences our *belief* (initially, Poisson(4)) about the distribution of the unknown value N.

Example 5.25 ***Capture-Recapture.*** *Suppose there are N fish of a certain species in a lake, where N is unknown, but an estimate of N from a few years earlier is available. A number n_1 of the fish have been caught, tagged and released. A later sample of n_2 of the fish has been caught, of which X are observed to be tagged. Can we estimate N, supposing that an earlier estimate was 40, $n_1 = 10$, $n_2 = 15$, and $X = 2$?*

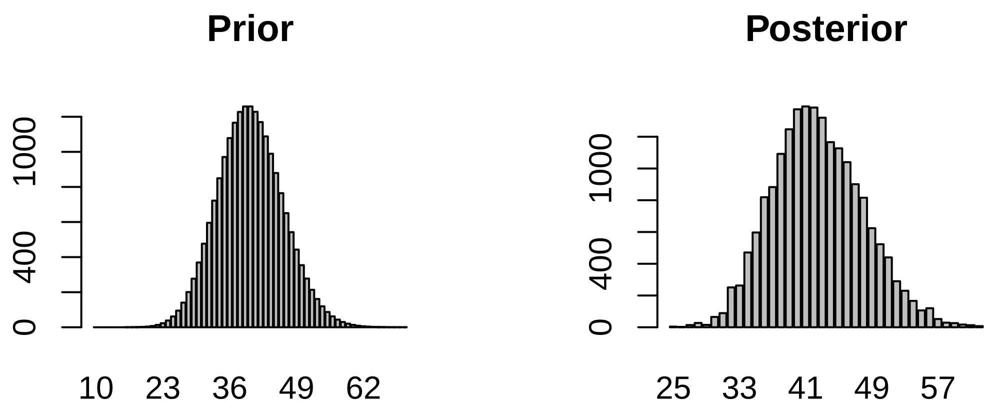
```r
posterior.fun <- function(n, x, n1, n2) {
    out <- 0
    if (x <= min(n1, n2)) out <- dpois(n, lambda = 40)*
            dhyper(x, n1, n-n1, n2)
    out
}
pi.fun <- function(i) {
    posterior.fun(i, x=2, n1=10, n2 = 15)
}
```

```r
N <- 20000
X <- numeric(N)
current.state <- 40  # initialize the Markov chain
for (n in 1:N) {
     i <- current.state
     P <- c(min(pi.fun(i-2)/pi.fun(i), 1),
            min(pi.fun(i-1)/pi.fun(i), 1),
            min(pi.fun(i+1)/pi.fun(i), 1),
            min(pi.fun(i+2)/pi.fun(i), 1))/6
```

```
      P0 <- 1 - sum(P)
      P <- c(P[1:2], P0, P[3:4])
      transition <- sample(seq(-2,2,1), size = 1, prob = P)
      current.state <- current.state + transition
      X[n] <- current.state
}
observedDist <- table(X[-c(1:1000)])
observedDist

##
##   25   26   27   28   29   30   31   32   33   34
##    4    2   14   27   15   65   89  251  263  471
##   35   36   37   38   39   40   41   42   43   44
##  597  819  883 1092 1247 1373 1391 1384 1320 1166
##   45   46   47   48   49   50   51   52   53   54
## 1127 1040  901  816  624  523  440  290  230  166
##   55   56   57   58   59   60   61   62
##  106  120   52   29   26   17   13    7
```

```
par(mfrow=c(1,2))
theoryDist <- 20000*dpois(10:70, lambda = 40)
names(theoryDist) <- 10:70
barplot(theoryDist, main = "Prior")
barplot(observedDist, main = "Posterior")
```



The posterior mode is 17*. Note also the cumulative posterior distribution of N:*

```
options(width=70)
round(cumsum(observedDist)/19000, 4)

##      25     26     27     28     29     30     31     32     33     34
## 0.0002 0.0003 0.0011 0.0025 0.0033 0.0067 0.0114 0.0246 0.0384 0.0632
##      35     36     37     38     39     40     41     42     43     44
## 0.0946 0.1377 0.1842 0.2417 0.3073 0.3796 0.4528 0.5256 0.5951 0.6565
##      45      46      47      48      49      50      51      52      53      54
## 0.7158 0.7705 0.8179 0.8609 0.8937 0.9213 0.9444 0.9597 0.9718 0.9805
##      55      56      57      58      59      60      61      62
## 0.9861 0.9924 0.9952 0.9967 0.9981 0.9989 0.9996 1.0000
```

From this output, we would judge that the probability that the number of fish is between 33 and 53 is approximately 90%. Note that this 90% credible interval is different from a 90% confidence interval, because it actually specifies an event with an actual probability level as opposed to the more nebulous "confidence level" attached to a confidence interval.

5.3.11 Using Built-In Software

One way to do MCMC in R is with the `metrop()` function in *mcmc* package (Geyer and Johnson, 2020). The syntax for its use is

```
    metrop(obj, initial, nbatch, blen = 1, nspac = 1,
       scale = 1, outfun, debug = FALSE, ...)
```

The main arguments are:

- `obj`: an R function which evaluates the unnormalized posterior distribution or the result of a previous call to this function.
- `initial`: the initial state of the Markov chain.
- `scale`: controls the proposal step size in the random walk used for the Markov chain.

We should also mention *R2WinBUGS* (Sturtz et al., 2005) and *rjags* (Plummer, 2022) which employ Gibbs sampling. Installation of *rjags* may not be completely straightforward. See `https://cran.r-project.org/web/packages/rjags/INSTALL` if you are installing the package from source.

Using Built-In Software

A simpler to use function, good for learning with, is `rwmetrop()` in the *LearnBayes* package (Albert, 2018).

```
 rwmetrop(logpost,proposal,start,m,...)
```

Arguments:

- `logpost`: function defining the log posterior density
- `proposal`: a list containing var, an estimated variance-covariance matrix, and scale, the Metropolis scale factor
- `start`: vector containing the starting value of the parameter
- `m`: the number of iterations of the chain

Using Built-In Software - Random Walk Metropolis-Hastings

Example:

- Suppose X is normally distributed with mean M and with known variance σ^2.
- We suppose M is a random variable having a normal distribution with parameter μ.
- We can use *Random Walk* Metropolis-Hastings to simulate from the posterior distribution of M, given the value of X.

Random Walk Metropolis-Hastings

- Random Walk Metropolis-Hastings is a version of the random walk where the jumps follow a continuous (actually, normal) distribution.
- The jump size is controlled by a single parameter: `scale`. Larger values of `scale` will give larger jumps.
- Too few jumps and too many large jumps will prevent the Markov chain from converging, so the scale needs to be chosen so that the proportion of *accepted proposals* is a moderate value, not too near 1% or 99%. Some people recommend 25%, but this is not always the best.
- An accepted proposal amounts to a transition in the Markov chain where a move is made; a rejected proposal is a transition where the Markov chain stays in the same state.

Using Built-In Software - Random Walk Metropolis-Hastings

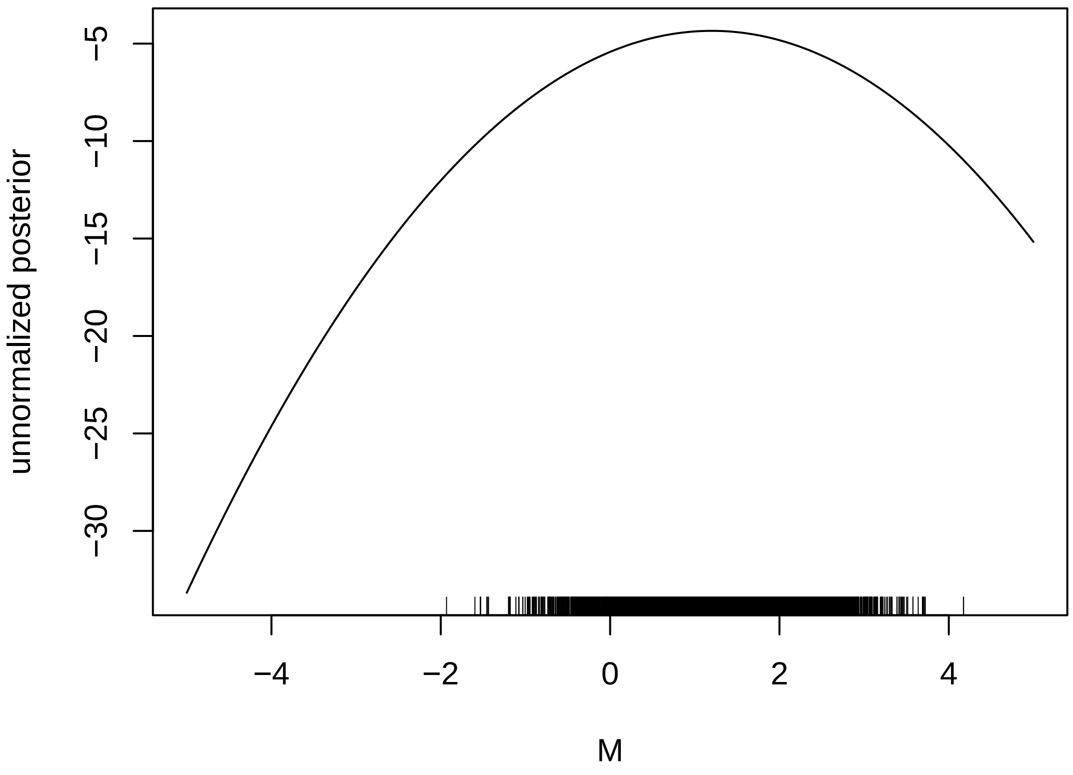
```r
logNormNorm <- function(M, datapar) {
    X <- datapar$data
    sigma2 <- datapar$sigma2
    Msd <- datapar$par
    loglike <- 0
    for (i in 1:length(X)) {
         loglike <- loglike + (dnorm(X[i], mean = M,
            sd=sqrt(sigma2), log = TRUE))
    }
    logprior <- dnorm(M, mean = 0,  sd = Msd, log = TRUE)
    return(loglike + logprior)
}
```

Using Built-In Software - Random Walk Metropolis-Hastings

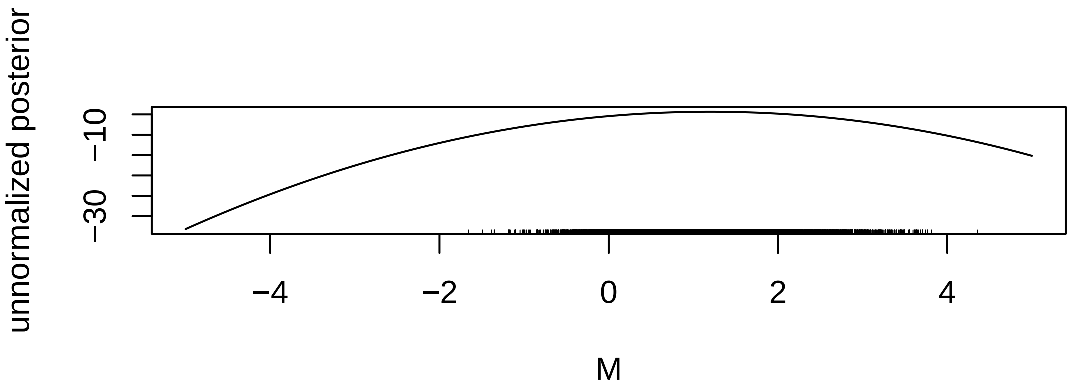
```r
library(LearnBayes)
data <- c(3.6)
start <- matrix(1, nrow = 1)
datapar <- list(data=data, par  = start, sigma2 = 2)
m <- 10000
varcov <- matrix(1, nrow=1)
proposal <- list(var=varcov, scale = 2)
s <- rwmetrop(logNormNorm, proposal, start, m, datapar)
s$accept

## [1] 0.4336

plot(seq(-5, 5, length=401), logNormNorm(seq(-5, 5, length=401),
    datapar=datapar), xlab=expression(M), type="l",
    ylab="unnormalized posterior")
rug(s$par)
```



Using Built-In Software - Random Walk Metropolis-Hastings



The rug plot identifies the simulated values from the posterior distribution.

Did We Choose the Right Scale?

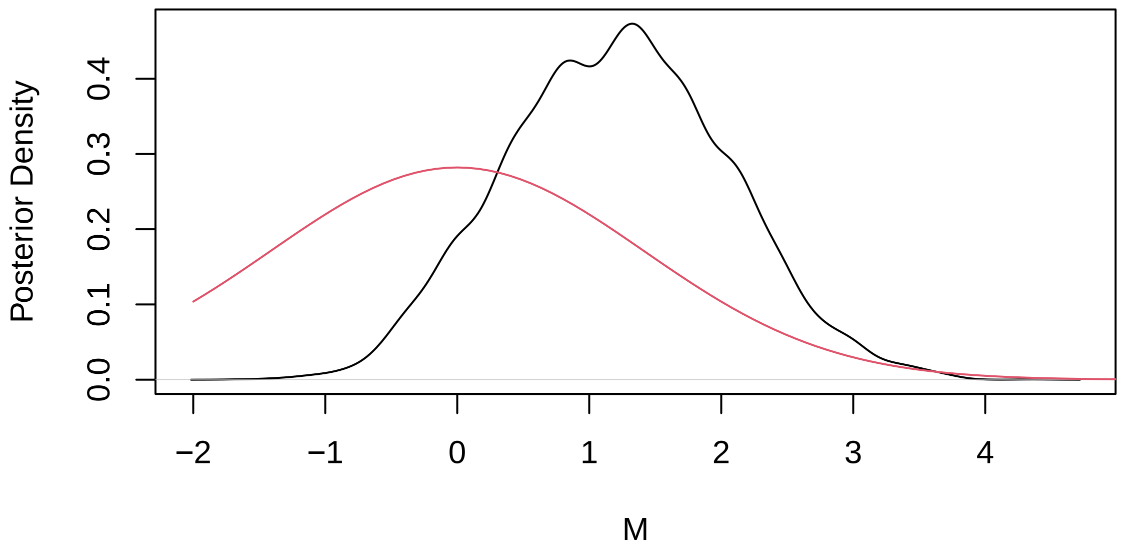
```
s$accept

## [1] 0.4407
```

Not bad.

Visualizing the Posterior Distribution

```
par(mar=c(4, 4, 1, 1))
plot(density(s$par), xlab="M", ylab="Posterior Density", main = " ")
curve(dnorm(x, mean = 0, sd = sqrt(datapar$sigma2)), -2, 6,
       col=2, add=TRUE)
```



the prior distribution is plotted in red.

Extending the Example to ANOVA

Recall ANOVA from our first class, and consider the chick weight data:

```
summary(chickwts)

##       weight                 feed
##  Min.   :108.0   casein   :12
##  1st Qu.:204.5   horsebean:10
##  Median :258.0   linseed  :12
##  Mean   :261.3   meatmeal :11
##  3rd Qu.:323.5   soybean  :14
##  Max.   :423.0   sunflower:12
```

Let's take M as the random variable representing the true mean weight for each type of feed. We can estimate σ^2 as follows:

```
chicks.lm <- lm(weight ~ feed,  data = chickwts)
MSE <- summary(chicks.lm)$sigma^2
MSE

## [1] 3008.554
```

This is the estimated variance of an individual chick weight. The variance of the average of all weights in a group can be estimated by dividing this by the group size.

We will also make some adjustments to the `logNormNorm()` function, since we know beforehand that the average weight is around 250, not 0.

5.3.12 Using Built-In Software - Random Walk Metropolis-Hastings

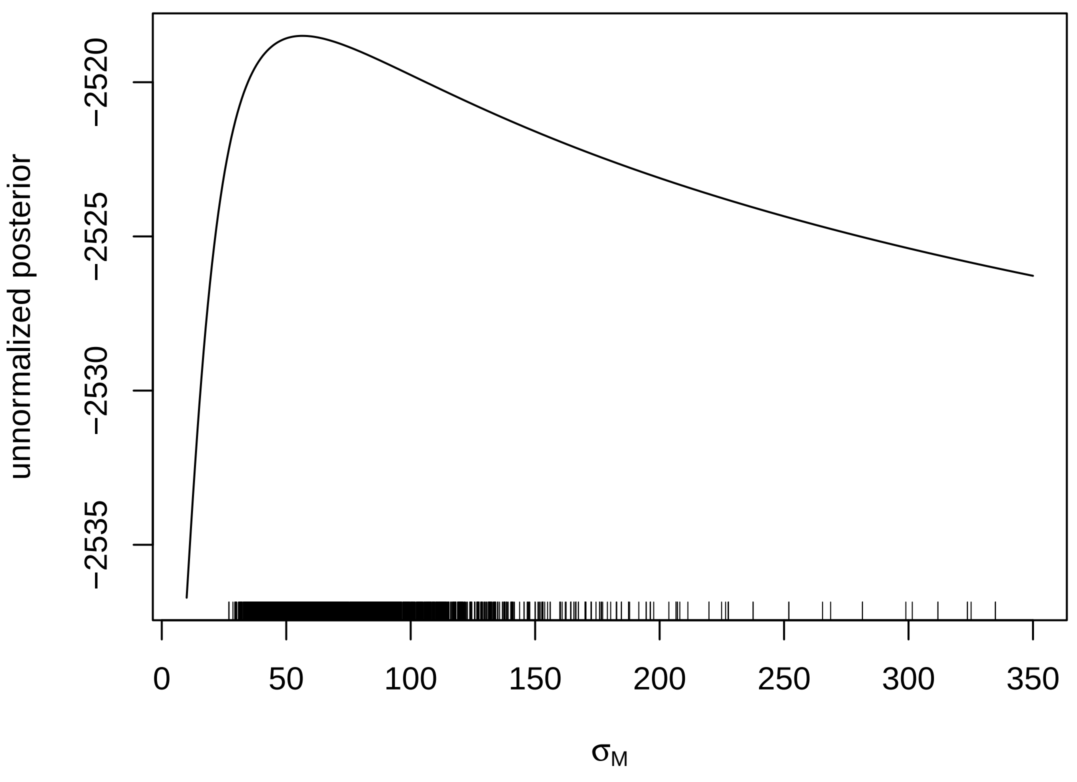
```
logNormExp <- function(Msd, datapar) {
    X <- datapar$data
    mu <- datapar$mu
    sigma2 <- datapar$sigma2
    lambda <- datapar$lambda
    n <- datapar$n
    loglike <- 0
    for (i in 1:length(X)) {
         loglike <- loglike + (dnorm(X[i], mean = mu,
            sd=sqrt(sigma2/n[i] + Msd^2), log = TRUE))
    }
    logprior <- sum(dexp(abs(Msd), rate = lambda,
                   log = TRUE))
    return(loglike + logprior)
}
```

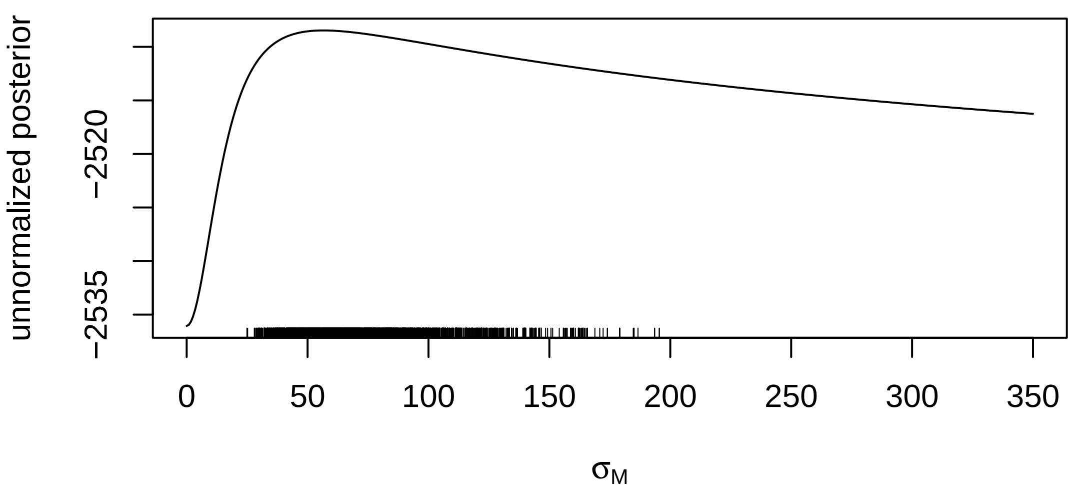
Applying MCMC to the Chick Weights Data

We will work with the average of the chick weights, giving us 6 data points, the ith one having conditional mean M and conditional variance estimated by MSE$/n_i$, where n_i is the number of chicks on diet i.

```
data <- sapply(split(chickwts$weight,
       chickwts$feed), mean)
n <- sapply(split(chickwts$weight,
       chickwts$feed), length)
start <- matrix(150,  nrow=1)
datapar <- list(data=data, mu = 261,  sigma2 = MSE, lambda = .005,
      par=start,  n = n)
m <- 10000
varcov <- matrix(1, nrow=1)
proposal <- list(var=varcov, scale = 100)
s <- rwmetrop(logNormExp, proposal, start, m, datapar)
plot(seq(10, 350, length=401), logNormExp(seq(10, 350, len=401),
    datapar=datapar), xlab=expression(sigma[M]), type="l",
    ylab="unnormalized posterior")
rug(s$par)

## Warning in rug(s$par):  some values will be clipped
```


unnormalized posterior
−2520
−2525
−2530
−2535
0
50
100
150
200
250
300
350
σ_M


unnormalized posterior
−2520
−2535
0
50
100
150
200
250
300
350
σ_M

Did We Choose the Right Scale?

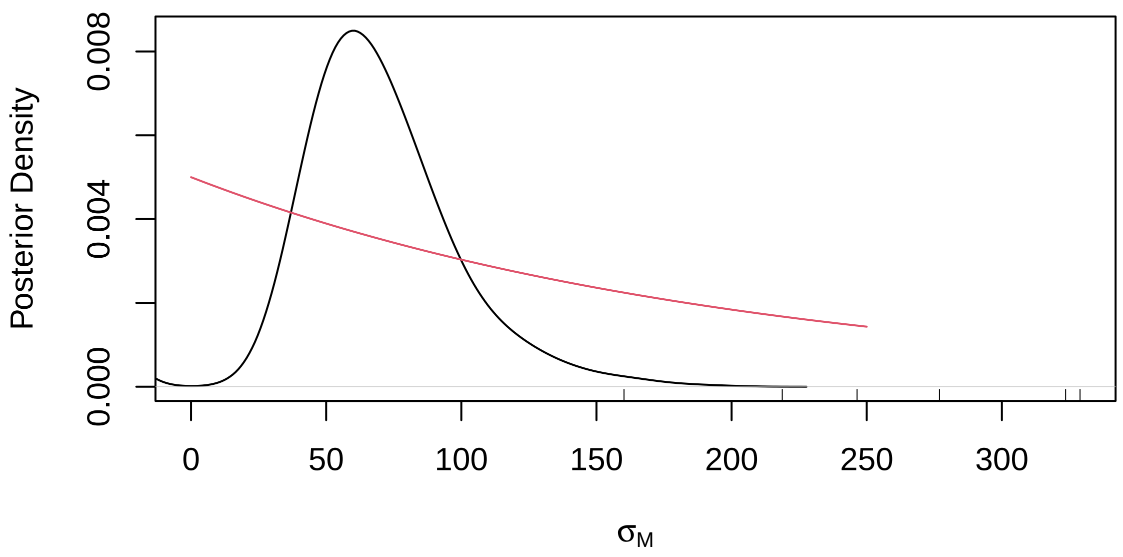
```
s$accept

## [1] 0.3477
```

Not bad.

Visualizing the Posterior Distribution

```
par(mar=c(4, 4, 1, 1))
plot(density(s$par), xlab=expression(sigma[M]),
   ylab="Posterior Density", xlim=range(c(0, data)),
   main = " ")
curve(dexp(x, rate = datapar$lambda) ,
    0, 250, add=TRUE, col=2)
rug(data)
```

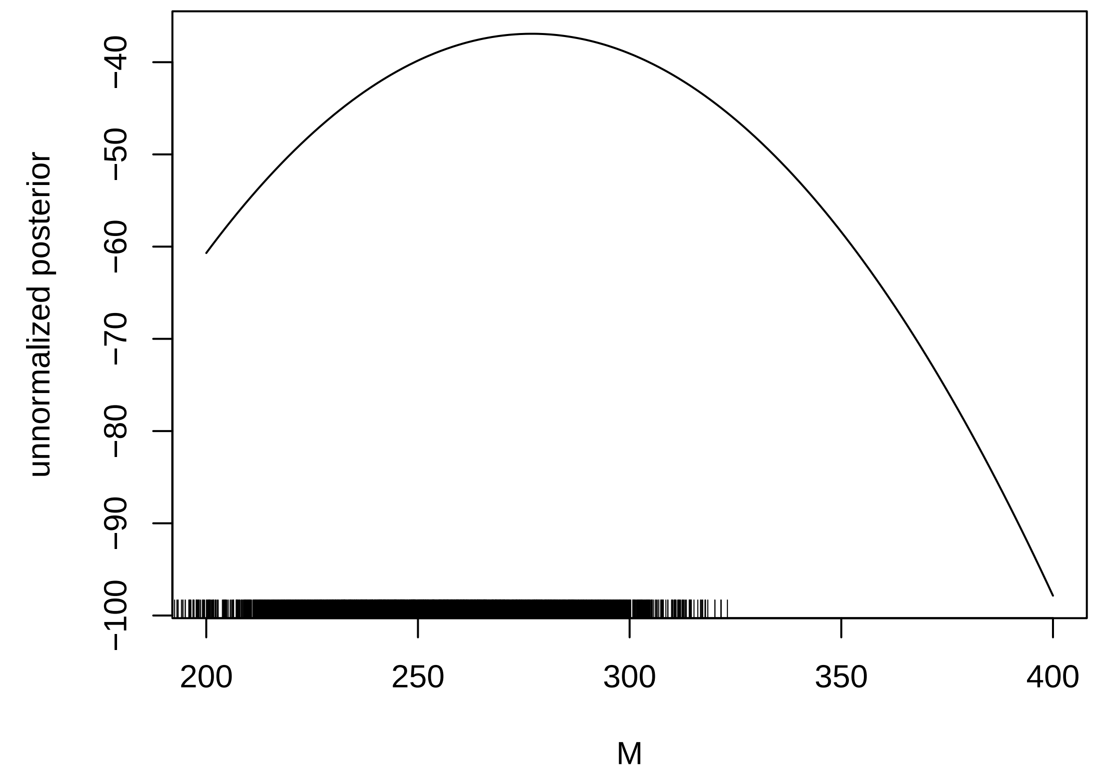


The prior density is plotted in red.

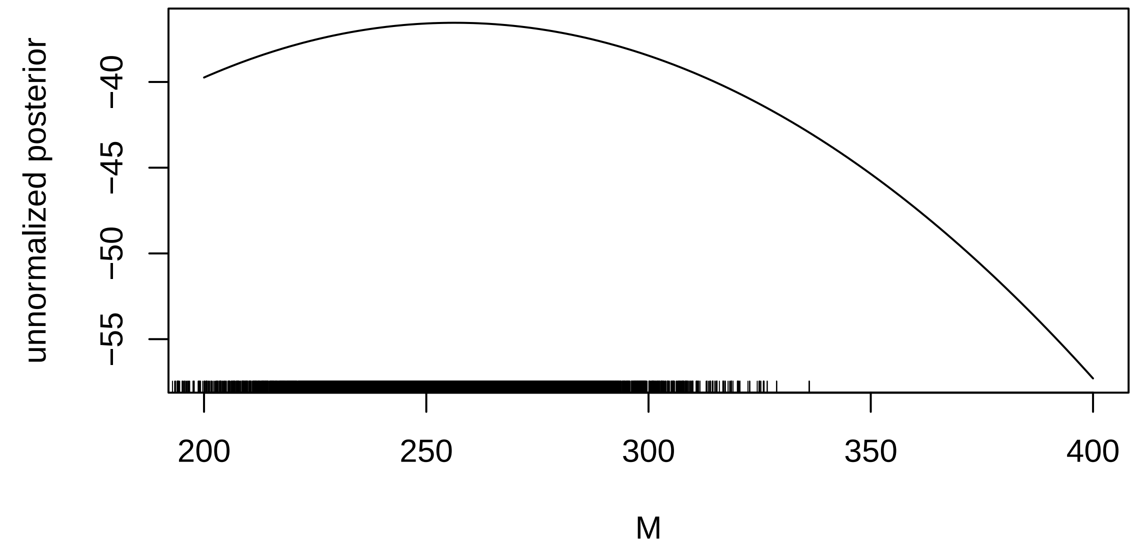
Compare with a Null Reference Data Set

```
data <- rnorm(6, mean = 250, sd = sqrt(MSE/n))
start <- matrix(250, nrow=1)
datapar <- list(data=data,par=start, sigma2 = MSE, n = n,
       mean = 250, sd = 50)
m <- 10000
varcov <- matrix(1, nrow=1)
proposal <- list(var=varcov, scale = 25)
s <- rwmetrop(logNormNorm, proposal, start, m, datapar)
plot(seq(200, 400, length=401), logNormNorm(
        seq(100, 500, len=401),
    datapar=datapar), xlab=expression(M), type="l",
    ylab="unnormalized posterior")
rug(s$par)
```

```
## Warning in rug(s$par):  some values will be clipped
```



```
## Warning in rug(s$par):  some values will be clipped
```



Did We Choose the Right Scale?

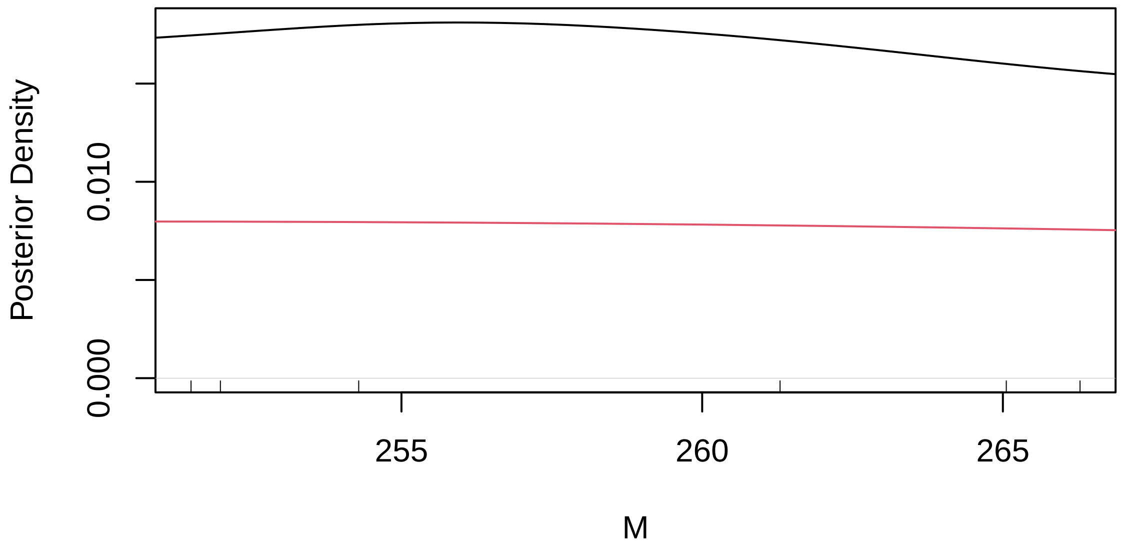
```
s$accept

## [1] 0.6788
```

Not bad.

Visualizing the Posterior Distribution

```
par(mar=c(4, 4, 1, 1))
plot(density(s$par), xlab="M", ylab="Posterior Density",
   xlim=range(data), main = " ")
curve(dnorm(x, mean=datapar$mean, sd=datapar$sd),
    200, 400, add=TRUE, col=2)
rug(data)  # the simulated data points (averages)
```



The prior density is plotted in red.

Exercises

1. Use conventional or crude simulation to generate 100000 independent observations on the stationary distribution for the 7-state random walk with reflecting barrier considered in Example 5.16. Calculate the relative frequencies for each state and compare with the theoretical distribution. How many digits are accurate? Explain why the sample generated in this exercise cannot be used to obtain the runlength distribution for visits avoiding State 5?

2. The following is a transition matrix for a Markov chain $X_1, X_2, \ldots$ on the state space $\{1, 2, 3\}$:

```
##        [,1]  [,2] [,3]
## [1,] 0.500 0.250 0.25
## [2,] 0.125 0.625 0.25
## [3,] 0.050 0.100 0.85
```

   (a) Find $P(X_5 = 1 | X_1 = 2)$.

(b) Is P regular? If so, find the limiting stationary distribution of the Markov chain.

(c) Is this Markov chain time-reversible? Explain in detail.

3. Consider the following transition matrix for a Markov chain $X_1, X_2, \ldots$ on the state space $\{0, 1, 2, 3, 4, 5\}$, where X_j represents the number of individuals who get on a bus at a stop at time j:

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  0.9 0.10 0.00 0.00 0.00  0.0
## [2,]  0.1 0.85 0.05 0.00 0.00  0.0
## [3,]  0.0 0.10 0.80 0.10 0.00  0.0
## [4,]  0.0 0.00 0.10 0.80 0.10  0.0
## [5,]  0.0 0.00 0.00 0.05 0.85  0.1
## [6,]  0.0 0.00 0.00 0.00 0.10  0.9
```

(a) Find $P(X_3 = 3|X_1 = 2)$.

(b) Is this Markov chain time-reversible? Explain in detail using the following vector q.

```
q

## [1] 0.2 0.2 0.1 0.1 0.2 0.2
```

(c) Find the limiting stationary distribution of the Markov chain, and find the long-run expected value of the number of individuals who get on the bus at the given stop.

4. Suppose $\pi = [.11\ .22\ .03\ .04\ .05\ .11\ .22\ .03\ .04\ .15]$, and consider the time-reversible Markov chain:

$$P_{i,j} = \frac{1}{3} \min\left(\frac{\pi_j}{\pi_i}, 1\right), \text{ for } j = i-1, i+1$$

and $P_{i,j} = 0$, for $|j - i| > 1$. $P_{i,i}$ is set to ensure that the row sums of P are 1.

Calculate the value of πP.

5. Refer to the above question, and suppose 20000 transitions of a Markov chain are simulated from the 4×4 version of P with $\pi = [1000\ 2000\ 3000\ 4000]$, using a starting vector chosen randomly from the distribution $[.1\ .2\ .3\ .4]$. How often would you expect your simulated Markov chain to visit state 3?

5.4 Hidden Markov models

Hidden Markov models have become an important application area of Markov chains. They provide more models with a high degree of flexibility while retaining enough simplicity to be reasonably interpretable. Many references are available on the subject, including the comprehensive treatment provided by Cappé et al. (2005).

5.4.1 A simple hidden Markov model

Suppose we observe data on a discrete random variable Y: $1, 0, 0, 0, 1, 1, 0$. We can model this as Bernoulli data, but we suspect there is some hidden dependence. So we choose to model it as

$$Y_j = B(0.25 + 0.5X_j)$$

where X_j is a Bernoulli random variable which is part of an underlying (hidden) Markov chain. Note that $P(Y_j = 1|X_j = 1) = .75$ and $P(Y_j = 1|X_j = 0) = .25$. These are called *emission probabilities*.

Since X_j is part of a Markov chain, we need a transition matrix. Since there are two hidden states, the transition matrix is 2×2, For example,

$$P = \begin{bmatrix} 1/3 & 2/3 \\ 2/3 & 1/3 \end{bmatrix}.$$

Together, the emission probabilities and the transition matrix make up a Hidden Markov Model (HMM).

Usually, we do not know the emission probabilities P_E or the transition probabilities P. These are usually estimated by maximizing the likelihood function:

$$L(P_E, P) = P(Y_1, Y_2, \ldots, Y_n).$$

Because the Y_j's are not independent, we need to be careful how to evaluate the likelihood.

What we can calculate easily:

$$P(Y_1, Y_2, \ldots, Y_n|X_1, X_2, \ldots, X_n) = \Pi_{j=1}^n (.25 + .5X_j)^{Y_j}(.75 - .5X_j)^{1-Y_j}.$$

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1)\cdots P(X_n|X_{n-1})$$

Taking the product of these, we can calculate:

$$P(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$$

We then have to sum over all combinations of the Xs to get the likelihood, a big job.

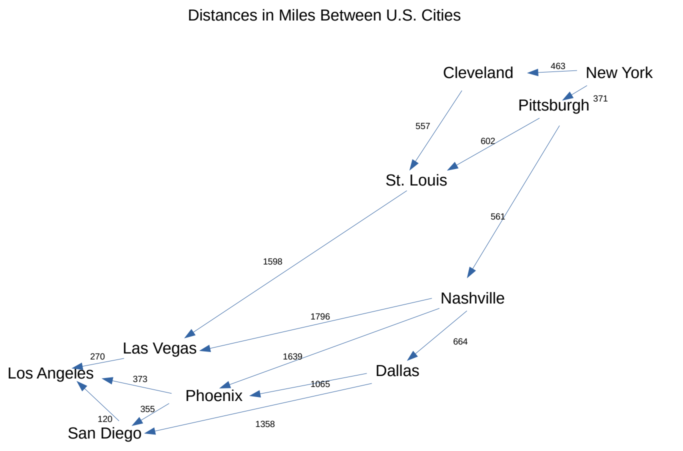
5.4.2 Dynamic programming

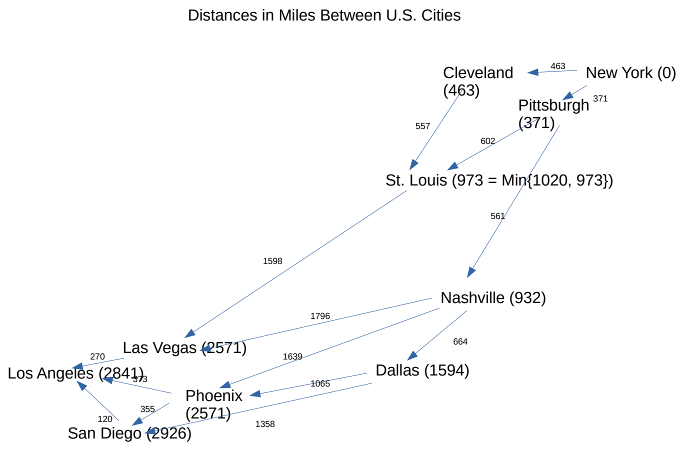
Dynamic programming is an optimization procedure due to Richard Bellman (early 1960's) which efficiently finds the minimum distance route through a network. Consider the problem of driving from Los Angeles to New York. There are many possible routes through the network of cities in between. Dynamic programming allows us to break the bigger network problem into a set of smaller network problems which can be solved recursively.

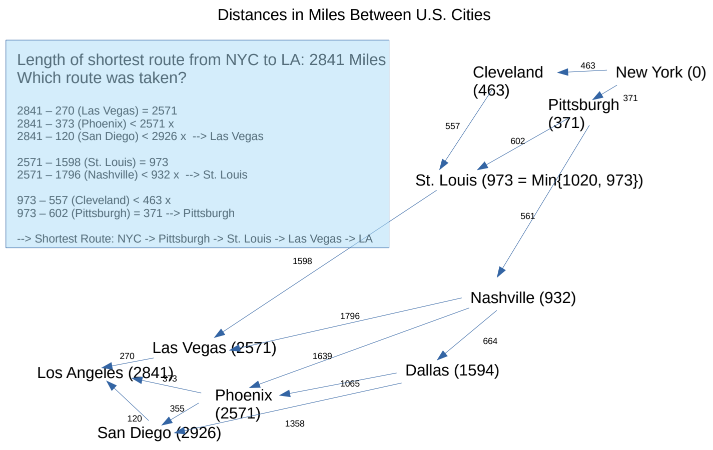
The basic Idea of the Viterbi algorithm

Dynamic programming can also be used to maximize quantities through a network. The Viterbi algorithm employs dynamic programming to find the mostly likely sequence of hidden Markov chain states which could give rise to the observed data.

Example 5.26 Find the Shortest Route from New York to Los Angeles







Length of shortest route from NYC to LA: 2841 Miles
Which route was taken?

2841 – 270 (Las Vegas) = 2571
2841 – 373 (Phoenix) < 2571 x
2841 – 120 (San Diego) < 2926 x --> Las Vegas

2571 – 1598 (St. Louis) = 973
2571 – 1796 (Nashville) < 932 x --> St. Louis

973 – 557 (Cleveland) < 463 x
973 – 602 (Pittsburgh) = 371 --> Pittsburgh

--> Shortest Route: NYC -> Pittsburgh -> St. Louis -> Las Vegas -> LA

Using the Viterbi path, we can calculate a kind of maximum likelihood estimate for the emission probability matrix and the transition matrix, based on the empirical proportions of time we observe the Y values, given the "observed" X values, and given the proportion of time we "observe" X values, given the preceding X values.

The Baum-Welch algorithm is an implementation of the Expectation-Maximization (EM) algorithm which calculates the exact maximum likelihood estimates of the parameters, given the Y observations.

Built-in software

The function `depmix()` provides a general purpose approach to fitting Hidden Markov Models to data. Details of its use goes beyond the scope of this book.. Instead, we focus on a simpler-to-use package, *HMM* (Himmelmann, 2022), which can be used to fit and simulate discrete-time discrete-state Hidden Markov Models.

We illustrate the use of the software for the Hidden Markov Model for which the transition matrix for the hidden 2 state Markov chain X is

$$P = \begin{bmatrix} 1/3 & 2/3 \\ 2/3 & 1/3 \end{bmatrix}$$

and the emission matrix for the probability distributions of the observed values Y, given X is

$$E = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{bmatrix}$$

The state space of the Markov chain is $S = \{0, 1\}$. The first row of the emission corresponds to the distribution of Y, given $X = 0$ and the second row corresponds to $X = 1$. We assume that the observed data are $Y = 1, 0, 0, 0, 1, 1, 0$.

We load the package, enter the information about the transition and emission matrices as well as the observed data.

```r
library(HMM)
```

```r
hmm <- initHMM(c("0","1"), c("0","1"),
        transProbs=matrix(c(1, 2, 2, 1)/3,2),
        emissionProbs=matrix(c(3, 1, 1, 3)/4, 2))
observations <- as.character(c(1, 0, 0, 0, 1, 1, 0))
viterbi <- viterbi(hmm, observations)
print(viterbi)

## [1] "1" "0" "1" "0" "1" "1" "0"
```

Using the `viterbi` function, we can calculate the Viterbi path, the most likely X values:

```r
viterbi <- viterbi(hmm, observations)
print(viterbi)

## [1] "1" "0" "1" "0" "1" "1" "0"
```

Training (estimating) the HMM from Y observations

Based on the preceding Viterbi sequence, we could get estimates of the transition matrix elements by calculating the proportion of the various transitions.

In the example, the Markov chain appears to have visited state 0 two times, not including the last value. The Markov chain entered state 1 both times, so we would estimate the P_{01} to be 1 and P_{00} as 0. Similarly, we would estimate P_{10} as $3/4$ and P_{11} as $1/4$.

We can also estimate the emission probabilities by estimating the distribution of Y for each value of X: $P(Y = 0|X = 0) = 2/3$ and $P(Y = 1|X = 0) = 1/3$. $P(Y = 0|X = 1) = 1/4$ and $P(Y = 1|X = 1) = 3/4$.

1. Begin with an initial guess as to the transition matrix and emission matrix.

2. Apply the Viterbi algorithm to the Y observations to obtain the most probable set of X observations, using the most recent estimates of the transition and emission matrices.

3. Use the Viterbi sequence of X's to estimate the transition matrix probabilities and the emission matrix probabilities.
4. Return to step 2, unless the transition and emission matrices have converged to within a given tolerance.

The function `viterbiTraining` is an implementation of this algorithm.
We now demonstrate this procedure on a simulated data set.

Example 5.27 *We simulate X's from a two state Markov chain, and use these to generate Y observations which take values 0, 1 and 2, according to different distributions, depending on the corresponding value of X. The transition matrix for the hidden Markov chain X:*

$$P = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.8 \end{bmatrix}$$

The emission matrix for the probability distributions of the observed values Y, given X:

$$E = \begin{bmatrix} 0.75 & 0.2 & 0.05 \\ 0.05 & 0.4 & 0.55 \end{bmatrix}$$

The first row corresponds to the distribution of Y, given $X = 0$ and the second row corresponds to $X = 1$.

```r
P <- matrix(c(.1, .9, .2, .8), nrow=2, byrow=TRUE)
E <- matrix(c(.75, .2, 0.05, .05, 0.4, .55), nrow=2, byrow=TRUE)
current.state <- 1; n <- 500
X <- numeric(n); Y <- numeric(n)
for (i in 1:n) {
    current.state <- sample(0:1, size = 1, prob = P[current.state+1,])
    X[i] <- current.state
    Y[i] <- sample(0:2, size = 1, prob = E[current.state+1,])
}
```

The first few simulated observations are

```r
Y[1:5]

## [1] 1 0 1 2 1
```

Fitting the HMM to the data, using an arbitrary starting guess for the emission matrix and the transition matrix.

```r
hmm <- initHMM(c("0","1"), c("0","1", "2"),
        transProbs=matrix(c(.5, .5, .5, .5),2),
        emissionProbs=matrix(c(.5, .4, .1, .8, .05, .15), 2))
observations <- as.character(Y)
simDataFit <- viterbiTraining(hmm, observations)
```

The output from the algorithm includes the estimated transition matrix and emission matrix:

```r
print(simDataFit$hmm$transProbs)  # transition matrix estimate

##      to
## from          0         1
##    0 0.1276596 0.8723404
##    1 0.2024691 0.7975309
```

```
print(simDataFit$hmm$emissionProbs)  # emission matrix estimate

##         symbols
## states 0         1         2
##      0 1 0.0000000 0.0000000
##      1 0 0.4408867 0.5591133
```

The estimated matrices are not far from the truth, but there is still considerable error. Note that the sample size is fairly large, and we are fitting a very simple model.

We can see how well our model did by comparing the original hidden Markov chain values with the most probable sequence that would be generated from the fitted Hidden Markov Model:

```
table(viterbi(simDataFit$hmm, observations), X)

##    X
##       0   1
##   0  69  25
##   1  31 375
```

5.4.3 Application to the Windspeed Data

Data preparation:

```
source("wind.R")
ws <- wind$h12  # Winnipeg noon hour windspeed (kmh)
wsCut <- cut(ws, c(-1e-10, 15, 22, 35, 45, 80))
levels(wsCut) # see what the cut function did

## [1] "(-1e-10,15]" "(15,22]"     "(22,35]"     "(35,45]"
## [5] "(45,80]"

levels(wsCut) <- c("N", "L", "M", "H", "E")
  # Nil, Low, Moderate, High, Extreme
WindStates <- factor(wsCut, ordered = TRUE)
```

Set up an initial guess for the HMM:

```
WindHMM <- initHMM(c("0","1","2"), levels(WindStates),
     transProbs=matrix(c(2, 1, 1, 1, 2, 2, 3, 3, 3)/6, 3),
     emissionProbs=matrix(c(4, 2, 0, 3, 2, 1, 2, 3, 2, 1,
   2, 3, 0, 2, 4)/10, nrow=3))
```

Viewing the transition matrix and emissions probabilities for the initial guess:

```
options(digits=3)
```

```
print(WindHMM)

## $States
## [1] "0" "1" "2"
##
## $Symbols
## [1] "N" "L" "M" "H" "E"
##
## $startProbs
##     0     1     2
## 0.333 0.333 0.333
##
## $transProbs
##     to
## from     0     1   2
##    0 0.333 0.167 0.5
##    1 0.167 0.333 0.5
##    2 0.167 0.333 0.5
##
## $emissionProbs
##       symbols
## states   N   L   M   H   E
##      0 0.4 0.3 0.2 0.1 0.0
##      1 0.2 0.2 0.3 0.2 0.2
##      2 0.0 0.1 0.2 0.3 0.4
```

Interpretation of the initial guess emissions probability matrix P_E:

$$P_E[i,j] = P(Y = j|X = i)$$

where X is the current state of the hidden Markov chain, and Y is the current observation.

e.g. $P_E[2,3] = 0.3$ so the probability of Moderate wind when in Markov Chain state 1 is 0.3.

Fitting the Hidden Markov Model by finding transition probabilities and emission probabilities to maximize the likelihood:

```
WindHMMFit <- viterbiTraining(WindHMM, WindStates)
```

Viewing the transition matrix and emissions probabilities for the fitted model:

```
print(WindHMMFit)

## $hmm
## $hmm$States
## [1] "0" "1" "2"
##
## $hmm$Symbols
## [1] "N" "L" "M" "H" "E"
##
## $hmm$startProbs
##      0      1      2
## 0.3333 0.3333 0.3333
##
## $hmm$transProbs
##     to
## from      0        1       2
##    0 0.9183 0.000000 0.08166
##    1 0.0000 0.976000 0.02400
##    2 0.8311 0.004444 0.16444
##
```

```
## $hmm$emissionProbs
##       symbols
## states     N      L      M      H      E
##      0 0.347 0.2368 0.4162 0.0000 0.0000
##      1 0.096 0.2640 0.6400 0.0000 0.0000
##      2 0.000 0.0000 0.0000 0.7778 0.2222
##
##
## $difference
##  [1] 1.58807 0.48527 0.40793 0.34788 0.26000 0.08013 0.06447
##  [8] 0.06521 0.10642 0.07139 0.11167 0.12598 0.07445 0.02481
## [15] 0.03118 0.00000
```

Simulate from the fitted model:

```
WindSim <- simHMM(WindHMMFit$hmm, length(WindStates))
```

We have simulated the same number of observations as in the original data set.

5.4.4 Application to the Windspeed Data

Compare run lengths of the various states:

```
table(rle(as.character(WindStates)))

##        values
## lengths    E    H    L    M    N
##       1  133  409 1017 1028 1031
##       2    7   49  217  410  307
##       3    1    6   37  175  115
##       4    0    0   13   74   39
##       5    0    0    4   28   22
##       6    0    0    3   14    8
##       7    0    0    0    2    6
##       8    0    0    1    3    4
##       9    0    0    0    1    2
```

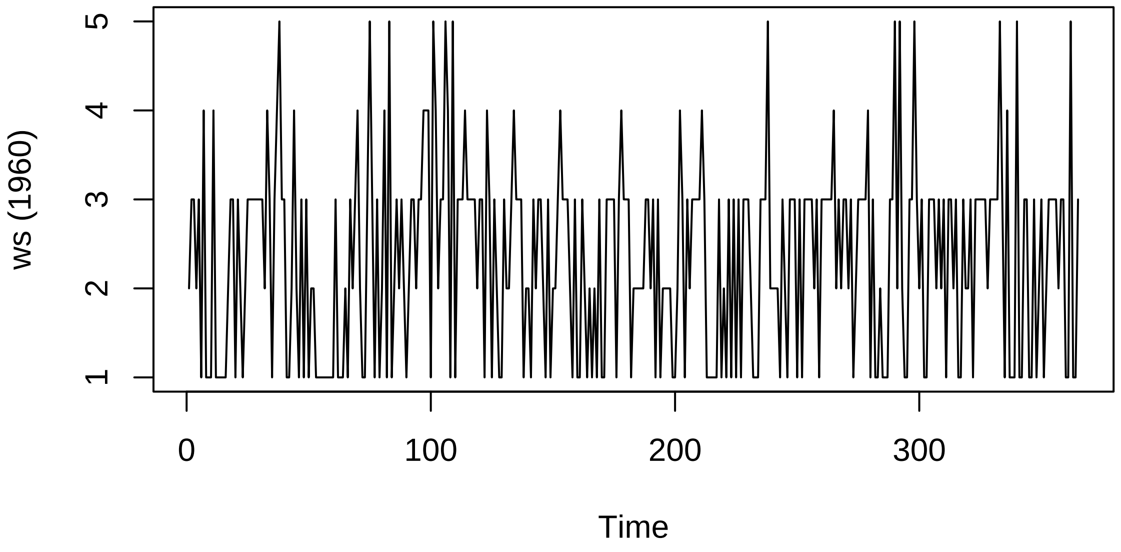
```
table(rle(WindSim$observation))

##        values
## lengths    E    H    L    M    N
##      1   146  371 1039 1095 1096
##      2     5   64  225  425  345
##      3     0    5   52  156  128
##      4     0    1    7   62   28
##      5     0    0    1   30   13
##      6     0    0    1    9    4
##      7     0    0    0    4    2
##      8     0    0    0    2    0
##      9     0    0    0    1    0
##      10    0    0    0    1    0
```

We have use the `rle()` (Run Length Encoding) function which computes the lengths and values of runs of equal values in a vector.

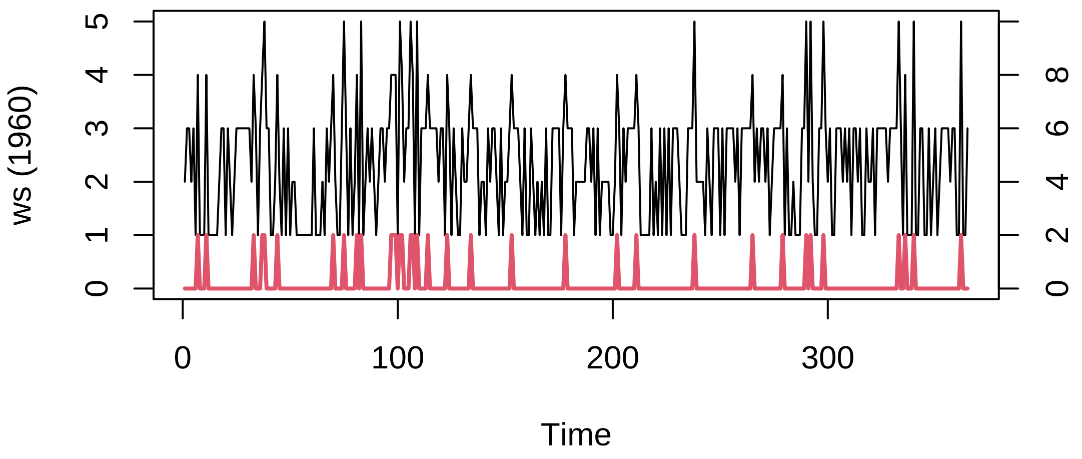
One year of the original data:

```r
par(mar=c(4, 4, 1, 1))
ts.plot(WindStates[1:365], ylab="ws (1960)")
```



Including information on the most likely hidden states:

```r
HiddenStates <- viterbi(WindHMMFit$hmm, WindStates[1:365])
par(mar=c(4, 4, 1, 4))
ts.plot(WindStates[1:365], ylab="ws (1960)", ylim=c(0, 5))
lines(1:365, as.numeric(HiddenStates)/2, col=2, lwd=2)
axis(side = 4, at=0:5, labels=seq(0,10,2))
```



Exercises

1. Suppose $Y_1, Y_2, \ldots$ are the outcomes of a Hidden Markov Model where we believe the transition matrix for the hidden Markov chain on the state space $\{1, 2, 3\}$ is

```
##      [,1] [,2] [,3]
## [1,]  0.1  0.2  0.7
## [2,]  0.8  0.0  0.2
## [3,]  0.7  0.3  0.0
```

and we think the Emission matrix is given by

```
##      [,1] [,2] [,3] [,4]
## [1,]  0.1  0.1  0.8  0.0
## [2,]  0.2  0.2  0.6  0.0
## [3,]  0.3  0.1  0.4  0.2
```

Use the Viterbi training algorithm to update the estimate of the transition matrix as well as the emission matrix based on the following observations:

```
##  [1] 3 1 2 3 2 1 3 3 3 3 2 3 1 3 3 2 3 3 3 3
```

2. Refer to the Hidden Markov Model which is listed on the previous question.
   (a) Find the limiting stationary distribution for the hidden Markov chain.
   (b) Use the result from the preceding part to obtain the long run probability that the observed values Y would be in state 4.
   (c) Find $P(Y_2 = 4 | Y_1 = 4)$.

3. Consider the stock market data (UK's FTSE Index) in the 4th column of `EuStockMarkets`. In this exercise, we will fit a hidden Markov Model to the first 1000 observations and use the model to determine a reasonable price for a European Call Option.
   (a) Run the following codes to convert the data into a form that the *HMM* package can handle easily. That is, we will convert the continuous data of the FTSE index to log returns and then classify the values into one of 10 categories separated by the 10th, 20th, ..., 90th percentiles.

```
FTSE <- EuStockMarkets[,4]
n <- length(FTSE)
lFTSE <- log(FTSE)
logreturns <- diff(lFTSE)
logreturnsQ <- c(min(logreturns)*1.01,
     quantile(logreturns, (1:9)/10), max(logreturns)*1.01)
logreturnsCut <- cut(logreturns, logreturnsQ)
levels(logreturnsCut) <- 1:10
```

   (b) Construct a time series plot of the `FTSE` series. Note the values around the 1000th observation and at the last observation.
   (c) Construct a 4×4 transition matrix for the hidden Markov chain, and a 4×10 emissions matrix as follows:

```
P <- matrix(c(1:4, 4:1, 1, 4, 4, 1, 4, 1, 4, 1), nrow=4, byrow=TRUE)/10
PE <- matrix(c(1:10, 10:1, rep(5.5, 10), c(rep(1, 5), rep(10, 5))),
    nrow=4,  byrow=TRUE)/55
```

   (d) Load the *HMM* library and initialize the hidden Markov model.

(e) Fit the hidden Markov model to the first 1000 observations of the `logreturnsCut` data. Assign the fitted model to `hmmFit`.

(f) Execute the following code to simulate from the fitted model, convert it back to the original scale and use the simulated values to predict the last value of the FTSE time series.

```
nTraining <- 1000
N <- 1000
nTesting <- n - nTraining
Index859 <- numeric(N)
for (i in 1:N) {
    index <- as.numeric(simHMM(hmmFit$hmm, nTesting)$observation)
    Index859[i] <- exp(cumsum(c(log(FTSE[nTraining]), runif(nTesting,
           min=logreturnsQ[index], max=logreturnsQ[index+1]))))[nTesting]
}
```

Construct a histogram of the resulting simulated values which estimate the 1859th FTSE observation, and plot the actual 1859th observation using a rug plot. Are the simulated values realistic?

(g) Suppose at time 1000, we want to buy an option to buy into the FTSE index fund at time 1859. Let's suppose that the price of the fund is \$1 per unit, so when the fund is at 4000, we would be paying \$4000 to buy, and if it is at 5000, it would cost \$5000 to buy. By purchasing a European call option which expires on day 1859, we have the right to buy the index for a set price, say \$5000, even if the actual price is above that value. Of course, if the actual price is below \$5000, we would not exercise the right to buy, since we would be paying too much.

Based on our model, the price of such an option would be fairly priced using the expected value of the difference between the actual price and 5000 (if above 5000), or 0 (if below 5000):

```
mean((Index859 - 5000)*(Index859>5000))

## [1] 2278
```

The option costs over \$2000. Given that the price on day 1859 is only a little over \$5000, the purchaser of this option would have paid more than \$2000 to save around \$300. (Note, by the way, that we are ignoring interest in these calculations. Normally, that is factored in to the price.)

Revise the code of the previous part to find the price of the same European call option if purchased on day 1829 instead of day 1000. That is, we want to buy into the index at day 1859 for \$5000, 30 days in advance. Use the HMM fitted earlier to find the price of this option.

6
Simulation of Continuous Measurements

Figures 6.1 and 6.2 display histograms of measurements of North American river lengths, wind speeds, and German and U.K. daily stock returns. All four of the histograms display data of continuous numeric type, and all of the histogram shapes are most definitely not rectangular, so the distributions of these data sets are not uniform. Other probability models are needed to summarize these distributions.

The goal of this chapter is to show how independent uniform random numbers can be converted into independent random numbers that have distributions that resemble the distributions of actual measurements, such as those displayed in the two figures. In the process, we will see how different models for continuous data arise and can be used to make probabilistic predictions.

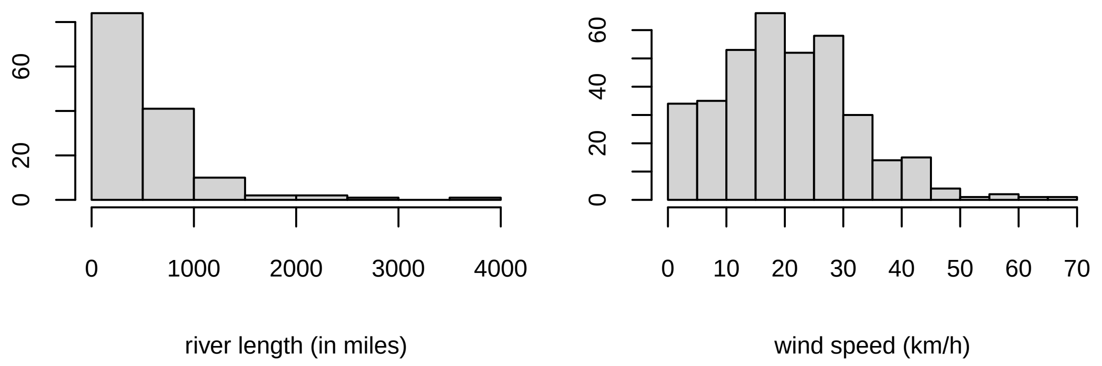


Figure 6.1: Left Panel: Histogram of lengths of long North American rivers. Right Panel: Histogram of noon hour wind speeds at Winnipeg International Airport

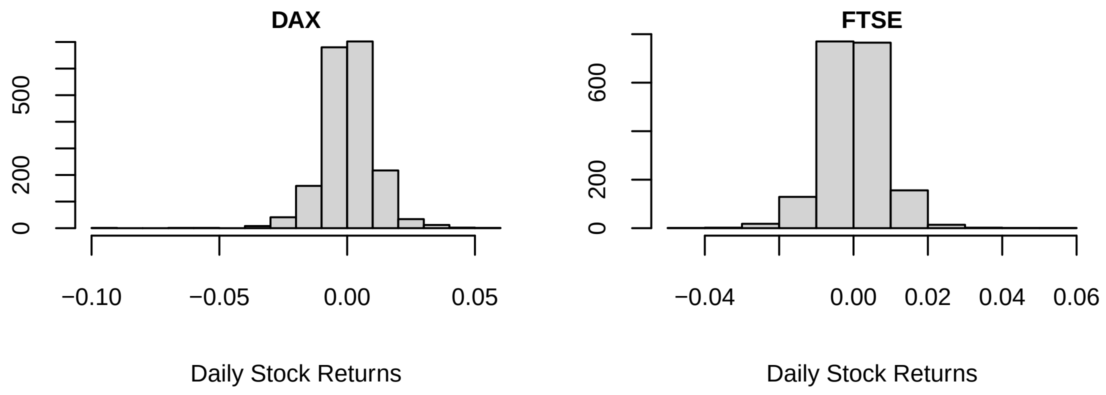


Figure 6.2: Histogram of stock returns for Germany's (DAX) and Britain's (FTSE) markets.

6.1 Uniform distributions

If U is a uniform random variable on the interval $[0, 1]$, we know that a histogram of a sample of size n coming from a population with the distribution of U will be rectangular with endpoints that correspond to the interval $[0, 1]$ which contains the possible values of U.

6.1.1 Linear transformations of uniform random variables

Multiplying a uniform random variable by a constant

If U is uniformly distributed on the interval $[0, 1]$, it is possible to show that if $V = bU$ for some positive constant b, then V will also be distributed as a uniform random variable but on the interval $[0, b]$.

Example 6.1 *Execution of the code below produces a sequence of 10000 uniform random numbers on the interval* $[0, 1]$ *which are plotted as a histogram in the left panel of Figure 6.3. In the right panel of the same figure, a histogram is displayed of the values multiplied by* $b = 2$*. The vertical axis of the second histogram has been controlled with the* `ylim` *argument so that the scale matches that of the first histogram. The number of histogram bins or breaks has also been controlled so that the binwidths are the same in both histograms.*

```
b <- 2; U <- runif(10000); V <- b*U
hist(U, main="", breaks=seq(0,1, length=10), ylim=c(0, 1200), xlim=c(-1, 2))
hist(V, main="", breaks=seq(0,2, length=20), ylim=c(0,1200), xlim=c(-1, 3))
```

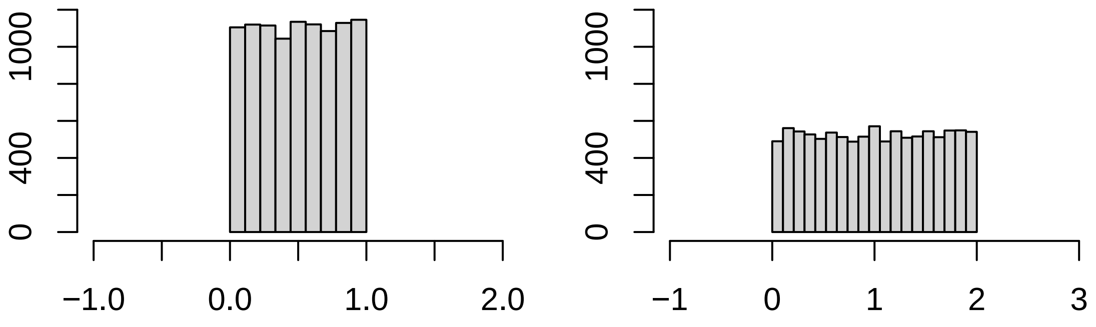


Figure 6.3: Histograms of 10000 simulated U and 2U values.

The figure shows that the distribution of $2U$ *is more spread out, but the rectangular distributional shape remains. The heights of the individual histogram bars are about half of the original histogram bar heights.*

Example 6.1 demonstrates that the areas of the histograms are the same, before and after multiplication by a constant. Both areas are equal to the sum of the bin counts: the total sample size, n. If all of the bin heights are taken to be the original bin counts divided by n, the histogram is called a relative frequency histogram. Such a histogram can be obtained by setting the `hist()` parameter `freq` to the value `FALSE`. Figure 6.4 shows the relative frequency histograms for the simulated uniform samples from Example 6.1. The total area of the bars of the histograms is 1, corresponding to the total probability that any of the V values would lie anywhere in the interval.

```
hist(U, main="", freq = FALSE, xlim = c(-1, 3))
polygon(c(0, 0, 1, 1), c(0, 1, 1, 0), lwd=2)
hist(V, main="", freq = FALSE,  xlim = c(-1, 3), ylim = c(0, 1))
polygon(c(0, 0, 2, 2), c(0, .5, .5, 0), lwd=2)
```

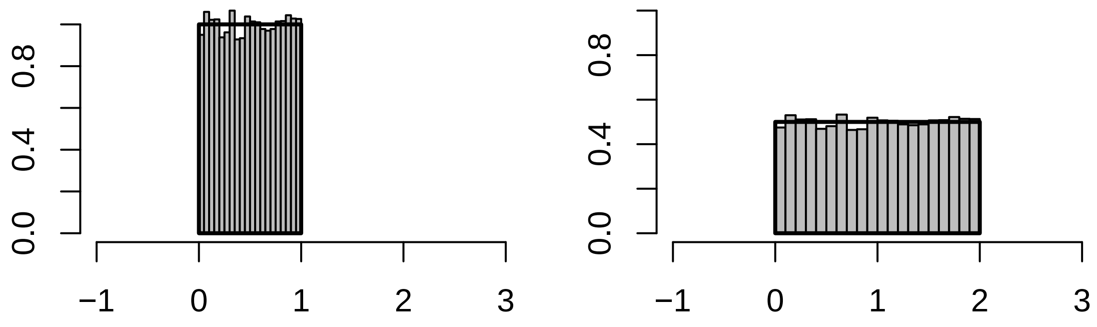


Figure 6.4: Relative frequency histograms of 10000 simulated U and 2U values with overlaid rectangles.

We have also overlaid each of the histograms in Figure 6.4 with rectangles having area 1 and having base widths of 1 and 2, respectively, which have been drawn with the `polygon()` function whose first two arguments give the horizontal and vertical coordinates of the rectangle vertices. The heights of these rectangles correspond to probability density, a concept we will return to in Section 6.1.6.

Adding a constant to a uniform random variable

If $W = bU + a$ for some constant a, then the distribution of W remains uniform, but on a different interval.

Example 6.2 *Execution of the code below produces a sequence of 10000 uniform random numbers on the interval* $[0, 1]$ *which are plotted as a histogram in the left panel of Figure 6.5. In the right panel of the same figure, a histogram is displayed of the values multiplied by* $b = 2$*, and then translated by* $a = 3$*. Note the use of the* `xlim` *argument to the* `plot()` *function which allows us to see the shape of the distribution over a larger interval. Again, the uniform probability density function curve has been overlaid, this time with* `min=3` *and* `max=5`.

```
b <- 2; a <- 3; U <- runif(10000); W <- b*U+a
hist(U, main="", freq = FALSE, xlim = c(-1, 6))
polygon(c(0, 0, 1, 1), c(0, 1, 1, 0), lwd=2)
hist(W, main="", freq = FALSE,  xlim = c(-1, 6), ylim = c(0, 1))
polygon(c(3, 3, 5, 5), c(0, .5, .5, 0), lwd=2)
```

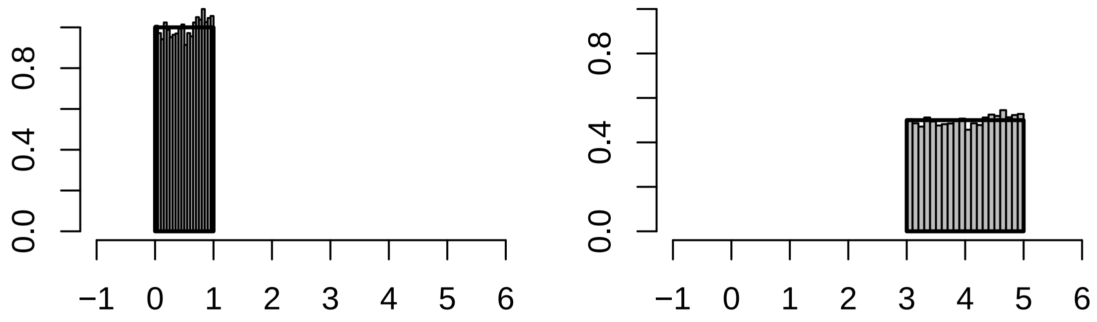


Figure 6.5: Relative frequency histograms of 10000 simulated U and $2U + 3$ values with overlaid rectangles.

This figure shows that after adding 3 to all values $2U$*, the resulting distribution is still uniform, but located on the interval* $[3, 5]$.

6.1.2 Cumulative distribution functions of uniform random variables

The cumulative distribution function (CDF) for a random variable V is a function of x, which is denoted by $F_V(x)$ and which returns the probability that V is less than or equal to x:

$$F_V(x) = P(V \leq x).$$

The total area between the horizontal axis and the uniform density rectangles graphed in each of the panels of Figure 6.6 is 1. The shaded areas correspond to the probabilities $P(U \leq 0.75)$ and to $P(W \leq 3.5)$, where U is uniform on $[0, 1]$, and W is uniform on $[3, 5]$.

The shaded area corresponding to $P(U \leq 0.75)$ is 0.75, and it should be evident that for any $x \in [0, 1]$, the area corresponding to $P(U \leq x)$ is x. Therefore,

$$F_U(x) = x, \quad x \in [0, 1].$$

If $x > 1$, the entire rectangle to the left of x would be shaded: $F_U(x) = 1$, for $x > 1$. If $x < 0$, then none of the rectangle would be shaded: $F_U(x) = 0$, for $x < 0$.

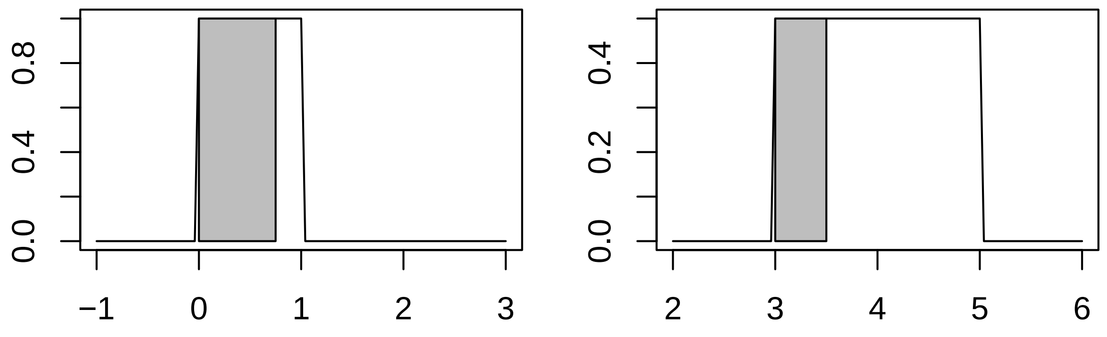


Figure 6.6: Shaded areas corresponding to $P(U \leq 0.75)$ and to $P(W \leq 3.5)$.

If $V = bU + a$ for some constant a, then we can show that the cumulative distribution function of V becomes

$$F_V(x) = P(V \leq x) = P(bU + a \leq x) = P(U \leq (x - a)/b) = \frac{x - a}{b}$$

when $x \in [a, a + b]$. When $x > a + b$, $F_V(x) = 1$, and when $x < a$, $F_V(x) = 0$.

The CDF of such a uniform random variable can be computed from the following function

```
cUnif <- function(x, a, b) {
    (x > a)*(x-a)/b + (x > (a + b))*(1 - (x - a)/b)
}
```

The built-in function `punif()` gives the same output where the parameters `a` and `b` are replaced by `min = a` and `max=a+b`, respectively.

Example 6.3 *Let $F_W(x)$ denote the CDF of $W = 2U + 3$. Then $F_W(-1) = 0$, $F_W(3.5) = .25$ and $F_W(5.5) = 1$. Compare these results with the output from execution of*

```
cUnif(-1, 3, 2); cUnif(3.5, 3, 2); cUnif(5.5, 3, 2)
punif(-1, 3, 5); punif(3.5, 3, 5); punif(5.5, 3, 5)
```

The CDF of V can be used to calculate the probability that V would take a value in any interval $(x, y]$, by subtraction

$$P(x < V \leq y) = F_V(y) - F_V(x).$$

Example 6.4 *Returning to Example 6.3, we can calculate the probability that W would take value between 3.5 and 4.5, from*

$$P(3.5 < W \leq 4.5) = F_W(4.5) - F_W(3.5) = 0.75 - 0.25 = 0.5.$$

or

```
punif(4.5, 3, 5) - punif(3.5, 3, 5)

## [1] 0.5
```

When plotted, the cumulative distribution function for a uniform random variable on $[a, a+b]$ appears as a continuous broken line curve having slope $1/b$ on the interval $[a, a+b]$, and slope 0, otherwise. Examples corresponding to uniform random variables on $[0, 1]$ and $[3, 5]$ appear in Figure 6.7.

```
curve(punif(x, 0, 1), -1, 6)
curve(punif(x, 3, 5), -1, 6)
```

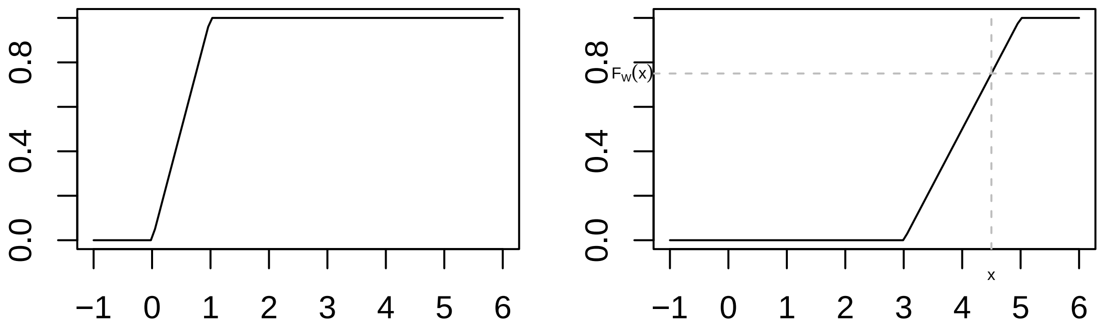


Figure 6.7: Graphs of the CDFs of U (left panel, $F_U(x)$) and $2U + 3$ (right panel, $F_W(x)$).

Values of probabilities of the form $P(V \leq x)$ can be read directly from such graphs. The dashed lines in the right panel indicate how to read off the probability that $P(W \leq x)$ for the given x value. In this case, we see that $F_W(4.5) = 0.75$.

6.1.3 Empirical distribution functions

Samples of data values can sometimes be usefully viewed as (finite) populations. We will see later that there is a technique called bootstrapping in which one takes a random sample or resample from a sample in order to perform certain calculations that might otherwise be quite difficult. Selecting a data point at random from a sample of distinct values $v_1, v_2, \ldots, v_n$ amounts to creating a discrete random variable, say V, which takes on any of the v values with equal probability. That is,

$$P(V = v_j) = \frac{1}{n}, \quad \text{for } j \in \{1, 2, \ldots, n\}$$

and 0, otherwise. The probability $P(V \leq x)$, for any real x, is then the proportion of v's that are less than or equal to x. This gives us the cumulative distribution function for V: $F_V(x)$, which is also called the empirical distribution function (EDF) for a sample.

To write $F_V(x)$ more concisely and practically, we need some very useful notation. Let $I(\text{statement})$ denote the indicator function of the truth of a given statement. In other words,

$$I(\text{statement}) = \begin{cases} 0 & \text{statement is FALSE} \\ 1 & \text{statement is TRUE} \end{cases}$$

If $v_j \leq x$, then $I(v_j \leq x) = 1$, so the total number of v_j's that are less than or equal to x must be $\sum_{j=1}^n I(v_j \leq x)$. Dividing this by n gives us the proportion that defined $F_V(x)$. That is,

$$F_V(x) = P(V \leq x) = \frac{1}{n}\sum_{j=1}^n I(v_j \leq x).$$

The empirical distribution function for a random sample $V_1, V_2, \ldots, V_n$ coming from a population with distribution function $F_V(x)$ is defined as

$$\widehat{F}_V(x) = \frac{1}{n}\sum_{j=1}^n I(V_j \leq x).$$

It is interesting to note that the indicator $I(V_j \leq x)$ is a random variable, and in particular, it is a Bernoulli random variable, for which the parameter $p = P(V_j \leq x)$. Since the V_j's are assumed to be independent, then the $I(V_j \leq x)$ Bernoulli random variables must also be independent. This means that the numerator of the empirical distribution function $\widehat{F}_V(x)$ is a binomial random variable with parameters n and p. Recall that the expected value of such a random variable is np, so

$$E[\widehat{F}_V(x)] = \frac{1}{n}[\sum_{j=1}^n I(V_j \leq x)] = \frac{np}{n} = p.$$

Since $p = F_V(x)$, we have that the expected value of the empirical distribution function is the cumulative distribution function of the population.

It is also possible to show, using the properties of the binomial distribution, that the variance of the empirical distribution function is $p(1-p)/n$, which decreases to 0 as n increases. A random variable with 0 variance is a non-random constant. Therefore, $\widehat{F}_V(x)$ tends to the non-random quantity $F_V(x)$, as n becomes infinitely large.

A graph of the empirical distribution function amounts to the same thing as a plot of a version of cumulative histogram for given data, corresponding to breakpoints taken at the data values themselves.

Comparing the EDF with the CDF

The `ecdf()` function can be used to calculate the empirical distribution function for a sample in R.

Example 6.5 *Figure 6.8 shows the uniform distribution CDF, together with a plot of the EDF based on a simulated sample of 100 uniform variates, obtained from the following code:*

```r
U <- runif(100)
plot(ecdf(U), col="grey", xlim=c(-1, 2), main="")
curve(punif(x), -1, 2, add = TRUE)
```

The grey colour has been used to display the EDF values, and the extended range $(-1, 2)$ *was used (by applying* `xlim`*) for display purposes.*

The figure shows that the EDF provides a reasonable estimate for the true CDF.

6.1.4 Cumulative histograms

The cumulative histogram provides another way to visualize univariate data sets. Like the EDF, it is an alternative to the conventional histogram which allows for direct comparison of the data with the cumulative distribution function $F_V(x)$. The bin counts of a cumulative histogram are obtained by successively adding the bin counts of

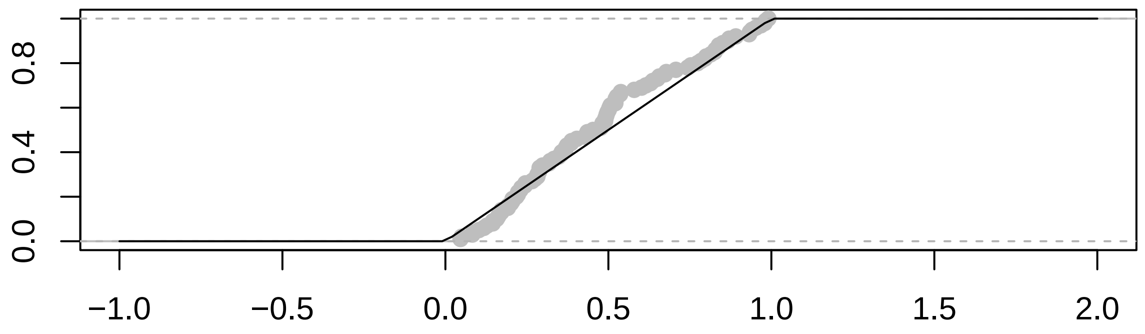


Figure 6.8: Histogram and cumulative histogram of 100 simulated U values. See Example 6.5.

the conventional histogram. The final bin will contain all of the counts, in other words, the size of the sample from which the original histogram was constructed. In order for the cumulative histogram to be directly comparable with the cumulative distribution function, it is necessary to divide each of the bin counts by the sample size (that is, the length of the vector containing the data).

The following function converts the conventional histogram counts into a relative cumulative histogram:

```
chist <- function(x, ...) {
    out <- hist(x, plot = FALSE)
        out$counts <- cumsum(out$counts)/length(x)
    plot(out, main="", xlab="", ...)
}
```

If we replace the third line of the code with `out$counts <- cumsum(out$counts)`, then we would be plotting the ordinary cumulative histogram.

Example 6.6 *Execution of the code below produces a sequence of 10000 uniform random numbers U on the interval $[0, 1]$ which are plotted as a cumulative histogram in the left panel of Figure 6.9. In the right panel of the same figure, a cumulative histogram is displayed of the values $V = 2U$. Note the use of the* `xlim` *argument to the* `plot` *function which allows us to see the shape of the graph of the function over a larger interval.*

```
chist(U, xlim=c(-1, 2));  curve(punif(x, 0, 1), from = -1, to = 2, add = TRUE)
chist(V, xlim=c(-1, 3)); curve(punif(x, 0, 2), from = -1, to = 3, add = TRUE)
```

6.1.5 *Quantiles of uniform random variables*

The preceding sections have been concerned with the problem of calculating the probability that a uniform random variable takes on a value less than some given quantity x. Inverting the question leads to the calculation of a quantile, the term being a generalization of the notion of percentile or quartile, and so on.

For a uniform random variable U on the interval $[0, 1]$, we have seen that $P(U \leq x) = x$ for any $x \in [0, 1]$. Thus, we can find any percentile of the distribution of U very easily. For example, the 57th percentile would simply be 0.57, which is found by recalling that this percentile is the value is larger than 57% of the rest of the distribution.

The pth quantile of U is the value q such that

$$F_U(q) = p. \tag{6.1}$$

Since $F_U(p) = p$ for all $p \in [0, 1]$, we see that the pth quantile of the distribution of U is necessarily p.

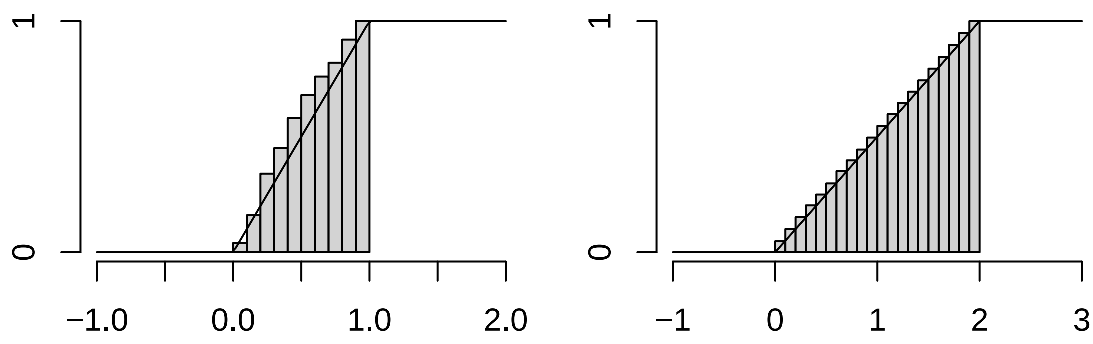


Figure 6.9: Cumulative histograms of 10000 simulated U and $V = 2U$ values with cumulative distribution functions overlaid.

For the general uniform random variable $V = a + bU$, we saw that the CDF was $F_V(x) = (x - a)/b$, for $x \in [a, a + b]$, so the pth quantile of V is the value q that satisfies

$$\frac{q - a}{b} = p,$$

that is, $q = bp + a$. For instance, the 75th percentile of the distribution of $W = 2U + 3$ is $q = 2(.75) + 3 = 4.5$.

The quantile function

Equation (6.1) implies that the pth quantile can be recovered from

$$q = F_U^{-1}(p)$$

where $F_U^{-1}(x)$ denotes the inverse function of $F_U(x)$. The function $F_U^{-1}(x)$ is called the quantile function for the distribution of U. The subsequent development of the section indicates that the quantile function for $V = a + bU$ is $F_V^{-1}(p) = bp + a$.

6.1.6 Probability density functions of uniform random variables

Earlier, we remarked that the height of the rectangle that overlaid a relative frequency histogram of simulated uniform data corresponded to probability density, In general, the height of histogram bars gives an indication as to which subregions of the whole interval have higher or lower probability of containing a randomly selected value.

In the case of a uniform random variable of the form $W = a + bU$, the probability density is flat, meaning that all subintervals of $[a, a + b]$ of the same length will have the same probability of occurrence. Since the height of the rectangle must be the total area divided by b, we define the probability density function for W as

$$f_W(x) = \begin{cases} \frac{1}{b}, & x \in [a, a + b] \\ 0, & \text{otherwise} \end{cases}$$

We have also seen in Figure 6.6 that if W is a uniform random variable on $[a, a + b]$, then $P(W \leq x)$ corresponds to the area of the portion of a rectangle of height $1/b$ to the left of the value x and to the right of the value a. Recalling that integrals have interpretations as areas under curves, we can write, for $x \in [a, a + b]$,

$$P(W \leq x) = \int_a^x \frac{1}{b} dy = \frac{x}{b} - \frac{a}{b},$$

and this interpretation agrees with our earlier assertion that

$$P(W \leq x) = \frac{x - a}{b}, \quad \text{for } x \in [a, a + b].$$

6.1.7 The expected value of a uniform random variable

The expected value of a uniform random variable W on $[a, a+b]$ is defined as

$$E[W] = \int_a^{a+b} x f_W(x)dx = \int_a^{a+b} x\frac{1}{b}dx = a + \frac{b}{2}.$$

This is simply the midpoint of the interval upon which the distribution is located.

Example 6.7 *When W is uniformly distributed on the interval $[3, 5]$, the expected value of W is $E[W] = 4$.*

The sample mean is an estimator for the expected value

We can use the `mean()` function to calculate the average of a collection of data values. The average is also called the sample mean.

Example 6.8 *We will simulate $n = 1000$ variates from a uniform distribution on the interval $[7, 12]$, and compute their sample mean:*

```
a <- 7; b <- 12
x <- runif(1000, min = a, max = b)
mean(x)

## [1] 9.49695
```

The average of this simulated sample is close to the theoretical mean of 9.5.

6.1.8 The variance of a uniform random variable

We have seen that a random variable is not associated with a single value, but rather with a distribution of values. The mean gives an indication of where those values might be located but does not contain any information about how spread out the distribution of values might be. The variance is often used as a measure of the spread of a distribution. Larger values indicate larger amounts of spread. A variance of 0 indicates no spread: the distribution is confined to a single value in this case.

The variance of a uniform random variable W on $[a, a+b]$ is defined as

$$\text{Var}(W) = \int_a^{a+b} (x - E[W])^2 f_W(x)dx = \int_a^{a+b} \left(x - a - \frac{b}{2}\right)^2 \frac{1}{b}dx = \frac{b^2}{12}.$$

As might be expected, the amount of variation in the distribution of W depends on b only, since this parameter uniquely specifies the width of the interval.

Example 6.9 *When W is uniformly distributed on the interval $[3, 5]$, the variance of W is $4/12 = 1/3$.*

The sample variance is an estimator for the theoretical variance

We can use the `var()` function to calculate the variance of a collection of data values. This is also called the sample variance, and it is really another kind of average: this time it averages[1] the squared distances of each data point from their sample mean. The larger this kind of average is, the greater the tendency for a data point to be far from the average, and the more spread out the distribution must be.

Example 6.10 *For the uniform sample simulated in Example 6.8, we can compare the sample variance with the theoretical variance as follows:*

[1]The denominator of the sample variance of n data points is $(n-1)$ instead of n, so that the expected value of the sample variance exactly matches the variance of the population from which the data points have been drawn.

```
var(x)

## [1] 2.10382
```

The variance of this simulated sample is close to the theoretical variance which is 25/12 = 2.083333.

6.2 Nonuniform distributions

The previous section contains a very thorough treatment of uniform random variables which could be viewed as a kind of "dress rehearsal" for the next section.

In the preamble to this chapter, we viewed the histograms of four data sets, none of which had the rectangular shape of a uniform distribution. To obtain other distributional shapes, it is necessary to use a nonlinear transformation of a uniform variable.

6.2.1 Nonlinear transformations of uniform variates: $V = g(U)$

We will see now that, although the uniform random variable does not often provide a useful model for real data, it is very often the building block for more useful models. In this first subsection, we will conduct a few small simulation experiments to see that effects of simple nonlinear transformations on uniform random variables.

Example 6.11 *If* $V = U^2$*, we obtain a relative frequency histogram as in Figure 6.10. The code to produce the figure is below.*

```
b <- 2; U <- runif(10000); V <- U^2
hist(U, freq=FALSE, main=""); hist(V, freq=FALSE, main="")
```

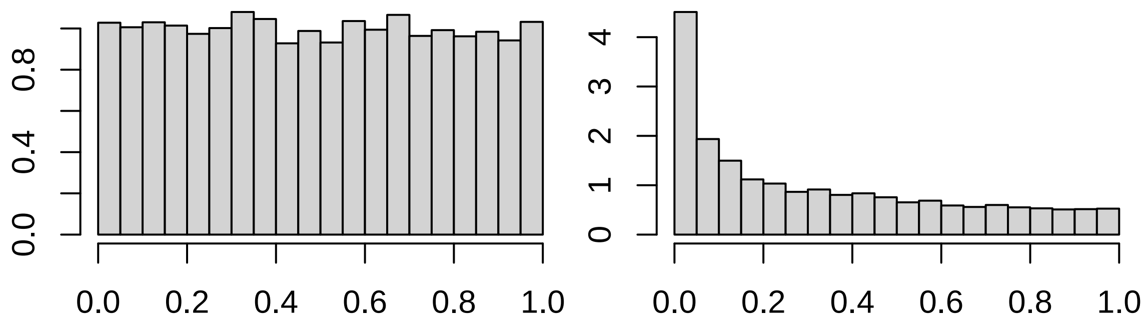


Figure 6.10: Relative frequency histograms of 10000 simulated U (left panel) and corresponding U^2 (right panel) values.

The resulting distribution is right-skewed, with many of the values near 0, with decreasing frequency as the values increase.

In general, we can simulate values from the distribution of any random variable V, when V can be expressed as a function of a uniform random variable. That is, if $V = g(U)$ for some function $g(x)$ that we can evaluate, then we can simulate values from the distribution of V.

Example 6.12 *Figure 6.11 shows the result of taking the natural logarithm of* U*:*

```
b <- 2; U <- runif(10000); W <- log(U)
hist(U, freq=FALSE); hist(W, freq=FALSE)
```

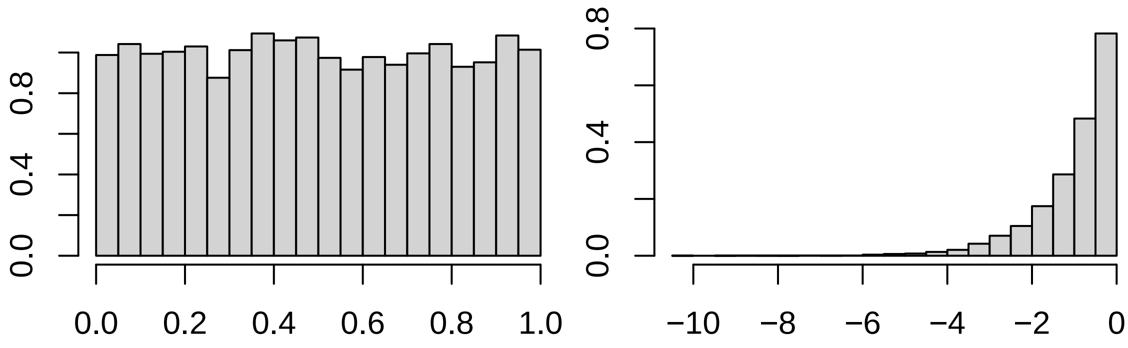


Figure 6.11: Histograms of 10000 simulated U and corresponding $\log(U)$ values.

This time, we see an exponential increase as the values approach 0, from the left. There may be occasions when such a distribution would be practical, but a more common situation, displayed in Figure 6.12, can be modelled by simply multiplying through by -1*:*

```
b <- 2; U <- runif(10000); X <- -log(U);
hist(U, freq=FALSE); hist(X, freq=FALSE)
```

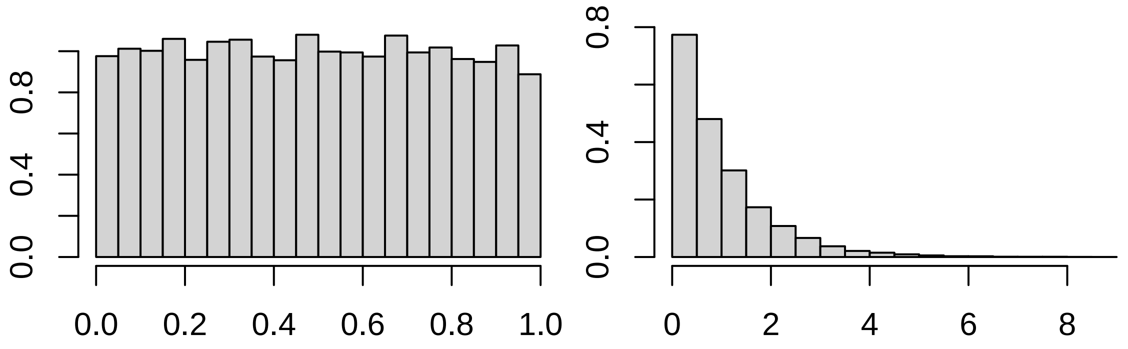


Figure 6.12: Histograms of 10000 simulated U and corresponding $-\log(U)$ values.

6.2.2 *Probability distributions of* $V = g(U)$

We have seen that the cumulative distribution function (CDF) of the uniform random variable U on the interval $[0, 1]$ is defined as

$$F_U(x) = P(U \leq x) = \begin{cases} x, & 0 < x \leq 1 \\ 1, & x > 1 \\ 0, & x \leq 0 \end{cases} \tag{6.2}$$

We have used the distribution function defined at (6.2) to find the cumulative distribution functions for other types of uniform random variables. In fact, we can also work out the distribution functions for other kinds of random variables, V, when we know their relationship with a uniform random variable, that is, when we know the function $g(x)$ for which $V = g(U)$.

Example 6.13 *If $V = U^2$, the distribution function is defined as*

$$F_V(x) = P(V \leq x).$$

We can use this definition to work out the form of the function $F_V(x)$. Since $V = U^2$, we must consider how to calculate the probability $P(U^2 \leq x)$. We know how to calculate probabilities involving U and not U^2, but we also know that when $U \geq 0$, then U is just the square root of U^2, and U^2 is less than or equal to some value x when, and only when, U is less than or equal to $\sqrt{x}$. Thus, the probabilities of these two events must be the same:

$$P(U^2 \leq x) = P(U \leq \sqrt{x}).$$

For $x > 1$, we see from (6.2) that $P(U \leq \sqrt{x}) = 1$. For $x \leq 0$, $P(U \leq \sqrt{x}) = 0$. Finally, for $x \in (0, 1]$, we have $P(U \leq \sqrt{x}) = \sqrt{x})$. Summarizing, we see that the CDF for V is

$$F_V(x) = P(V \leq x) = \begin{cases} \sqrt{x}, & 0 < x \leq 1 \\ 1, & x > 1 \\ 0, & x \leq 0 \end{cases} \tag{6.3}$$

As we saw for the uniform random variable case, the CDF of a random variable V can be used to calculate the probability that V would take a value in any interval $(a, b]$, by subtraction

$$P(a < V \leq b) = F_V(b) - F_V(a).$$

Example 6.14 *Returning to Example 6.13, we can compute the probability that V is in the interval* $[1/16, 1/4]$ *as follows:*

$$P(1/16 < V \leq 1/4) = F_V(1/4) - F_V(1/16) = \sqrt{1/4} - \sqrt{1/16} = 1/2 - 1/4 = 1/4.$$

Comparing the cumulative histogram with the CDF

Being able to compute the CDF, and to graph it, yields a way to check on simulation results to ensure that they are actually following the appropriate model, via the cumulative histogram. The argument above provides theoretical justification for doing this.

Example 6.15 *Returning to Examples 6.11 and 6.13, we use the following code to display the histogram of the simulated values from the CDF $F_V(x)$ on* $[0, 1]$, *as in Figure 6.13. The figure in the right panel is of the relative cumulative histogram of the simulated data, with the CDF overlaid.*

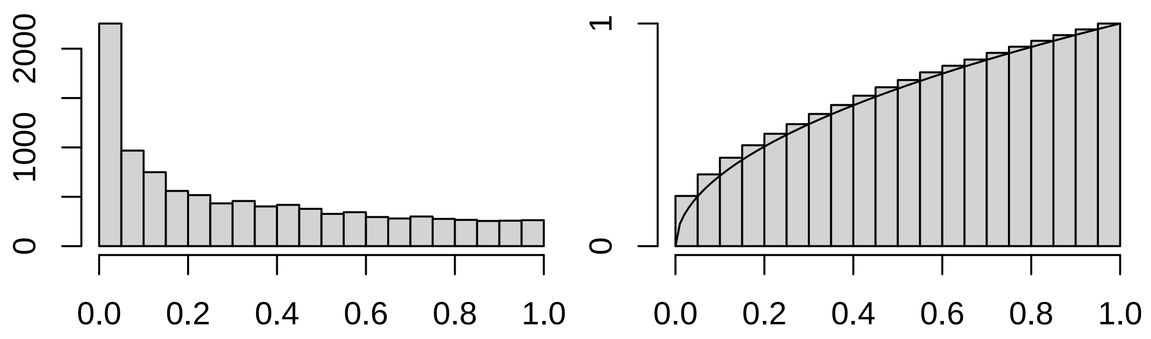


Figure 6.13: Histogram and cumulative histogram of 10000 simulated V values. See Example 6.11.

The right panel of the figure shows that the sample cumulative histogram and the true CDF match very well.

Example 6.16 *If $W = \log(U)$, we may obtain the distribution function by starting from:*

$$F_W(x) = P(W \le x) = P(\log(U) \le x).$$

This time, we note that $\log(U)$ will be less than or equal to x when, and only when, $U \le e^x$. Therefore,

$$F_W(x) = P(U \le e^x) = e^x$$

when $x < 0$, and $F_W(x) = 1$, when $x \ge 0$.

Figure 6.14 shows the histogram and relative cumulative histogram for data simulated from this model. The right panel contains the overlaid CDF which matches the cumulative histogram very well. The figure can essentially be obtained upon execution of the following code:

```
hist(W); chist(W);  curve(exp(x), -10, 0, add=TRUE)
```

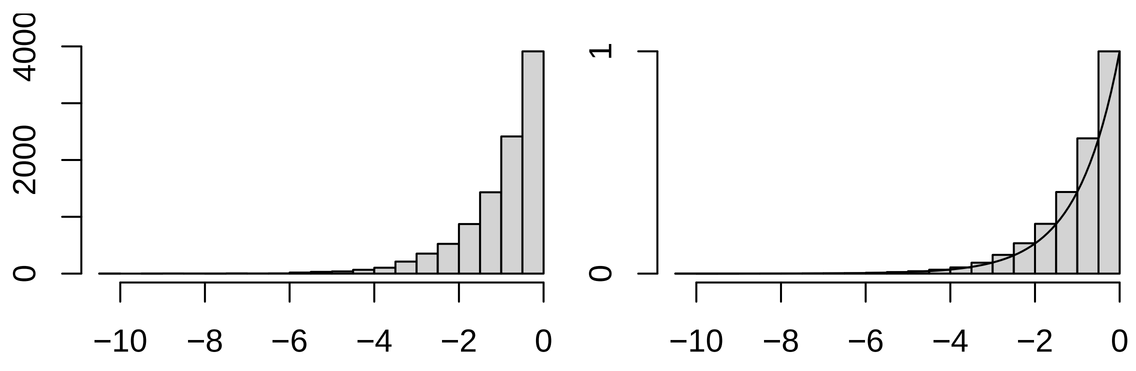


Figure 6.14: Histogram and cumulative histogram of 10000 simulated W values. See Example 6.12.

6.2.3 The CDF is the inverse of the simulation transformation

In the two examples of this section, the cumulative distribution function turned out to be the inverse function of the transformation that was originally applied to U. The square root is the inverse of the square for positive numbers, and the exponential function is the inverse of the natural logarithm. More generally, if $V = g(U)$, for some increasing function $g(x)$ (such as the square or the natural logarithm), the distribution function of V is given by

$$F_V(x) = P(V \le x) = P(g(U) \le x) = P(U \le g^{-1}(x)) = g^{-1}(x),$$

for $x \in [0, 1]$. If $g(x)$ is a decreasing function, such as $-\log(x)$, then the inequality above is flipped:

$$F_V(x) = P(V \le x) = P(g(U) \le x) = P(U \ge g^{-1}(x)) = 1 - g^{-1}(x),$$

for $x \in [0, 1]$.

6.2.4 Simulation with the inverse CDF

The final comments of the preceding section indicate that if we wish to simulate from the distribution $F_V(x)$, and we know that either $F_V(x) = g^{-1}(x)$ or $F_V(x) = 1 - g^{-1}(x)$, then we simply simulate uniform variates U on $[0, 1]$, and apply the transformation $V = g(U)$.

Observe that if $F_V(x) = g^{-1}(x)$, then $g(x) = F_V^{-1}(x)$. In other words, the transformation we seek is the inverse function of the distribution function itself. Thus, if you have a way of calculating the inverse function of the CDF, the following inverse-probability-transform method can be used to convert uniform numbers to the distribution you are targetting (that is, $F_V(x)$):

```
U <- runif(N) # simulate N uniforms
V <- Finv(U) # tranform to the target distribution
        # using the inverse of the CDF
V
```

Example 6.17 *Suppose V is a random variable with CDF $F_V(x) = \sin(x)$ for $x \in [0, \pi/2]$. Simulate 10000 random variates from this distribution.*

```
N <- 10000; U <- runif(N)
V <- asin(U) # this gives the arcsin, which is the inverse of sin
```

We can then use the following code to display the simulated values from the CDF $F(x) = \sin(x)$ on $[0, \pi/2]$, as in Figure 6.15.

```
hist(V); chist(V); curve(sin(x), 0, pi/2, add=TRUE)
```

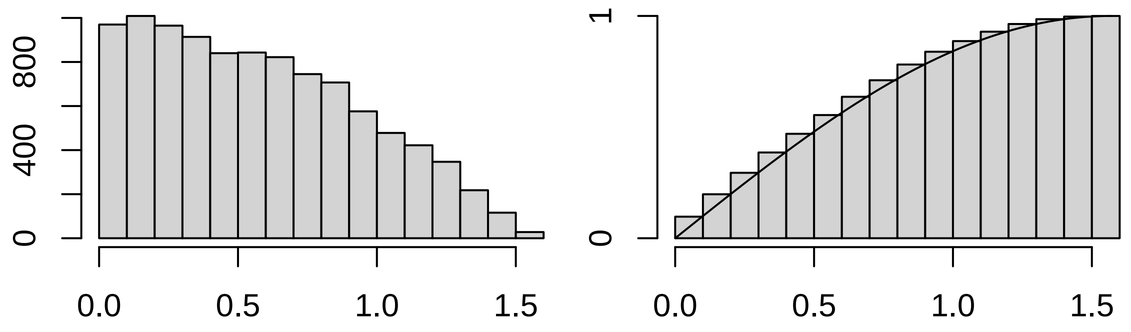


Figure 6.15: Histogram and cumulative histogram of 10000 simulated V values. See Example 6.17.

The left panel contains the histogram of the simulated data. The right panel contains the relative cumulative histogram of the simulated data, with the CDF overlaid. The sample cumulative histogram and the true CDF match very well.

Example 6.18 *Suppose V is a random variable with CDF $F_V(v) = 1 - 1/(v+1)^9$ for $x > 0$. Simulate 10000 random variates from this distribution. This distribution is sometimes called a Pareto distribution.*

The inverse of this CDF is found by solving

$$U = 1 - 1/(v+1)^9$$

for v:

$$1 - U = 1/(v+1)^9$$

so that

$$1/(1-U) = (v+1)^9$$

which implies

$$v = 1/(1-U)^{1/9} - 1$$

or

$$F_V^{-1}(U) = 1/(1-U)^{1/9} - 1.$$

Employing this in the inverse CDF method runs as follows:

```
N <- 10000; U <- runif(N)
V <- 1/(1-U)^(1/9) - 1
```

We use the following code to display the histogram of the simulated values from the CDF $F_V(x)$, as in Figure 6.16. The figure in the right panel is of the cumulative histogram of the simulated data, with the CDF overlaid.

```
hist(V); chist(V); curve((1-1/(x+1)^9), add=TRUE)
```

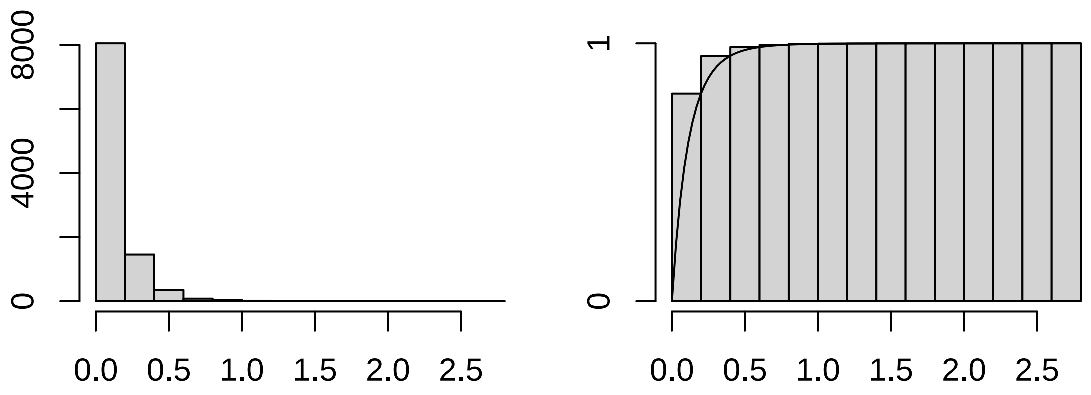


Figure 6.16: Histogram and cumulative histogram of 10000 simulated V values. See Example 6.18

Again, the sample cumulative histogram and the true CDF match very well. Notice that the sample has a right-skewed distribution.

Example 6.19 *An exponential random variable V has CDF*

$$F_V(x) = 1 - e^{-\lambda x}, \quad x \geq 0$$

and 0, otherwise. The constant λ must be positive, and it is often called the rate. To simulate N values from this distribution using the inverse CDF method, we find values X such that

$$U = 1 - e^{-\lambda X}$$

for uniform random numbers U. Solving this equation yields

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

This almost resembles the simulator described at the end of Example 6.12. The difference is the use of $1 - U$ instead of U in the argument of the log *function. In fact, both approaches will yield exponential random variables, since the distribution of $1 - U$ is the same as the distribution of U: for $x \in [0, 1]$,*

$$P(1 - U \leq x) = P(U \geq 1 - x) = 1 - (1 - x) = x = P(U \leq x).$$

6.2.5 Quantiles of distributions

In the case of the uniform distribution, quantiles are very straightforward to define. Quantiles can be defined for nonuniform distributions, but to avoid ambiguities, the definition involves a slight additional complication. The pth quantile of a distribution $F_V(x)$ is defined as the smallest value q such that $F_V(q) \geq p$. This quantile separates the lower pth proportion of a distribution from the upper $1 - p$th proportion.

Example 6.20 *The median corresponds to $p = 0.5$, so it is the smallest value q such that $F_V(q) \geq 0.5$. When V has an exponential distribution with rate λ, then the median q is the smallest value that satisfies*

$$1 - e^{-\lambda q} \geq 0.5.$$

Solving this inequality, we see that

$$q \geq \frac{-\log(0.5)}{\lambda}.$$

The smallest value of q, the median, is then

$$q = \frac{-\log(0.5)}{\lambda}.$$

In the previous example, we can see a connection between the median of the distribution of V and the median of the uniform distribution on $(0, 1)$: 0.5. More generally, if a value u has been generated from the uniform distribution on $[0, 1]$, the value of u will correspond to a particular percentile or quantile of the distribution of U, say q. (The numerical value of q will actually be identical to u, since the probability that a uniform random variable would be less than or equal to u is exactly u.)

Suppose that $g(u)$ is a monotonically increasing function of u. Then $g^{-1}(q)$ is also a monotonically increasing function of q. If values are simulated from the distribution of $V = g^{-1}(U)$, that is, from the CDF $F_V(x) = g(x)$, then the proportion of V's that are less than or equal to $g^{-1}(u)$ will be u, and the proportion greater than $g^{-1}(u)$ will be $1 - u$. Thus, $g^{-1}(u)$ is the uth quantile of the distribution of V.

Now, suppose that $u_1 < u_2 < \cdots < u_r$ are r ordered quantiles of the distribution of U. That is, $P(U \leq u_j) = u_j$, for $j = 1, 2, \ldots, r$. Then $g^{-1}(u_1) < g^{-1}(u_2) < \cdots < g^{-1}(u_r)$ must be the corresponding quantiles of the distribution of V.

Sample quantiles

Quantiles can be calculated for a sample $x_1, x_2, \ldots, x_n$ as well. The pth quantile of such a sample is the smallest value q such that the proportion of x_j's less than or equal to q q is at least p. Note that q is not necessarily unique, so a choice is often made, such as taking the midpoint of the range of possible values. The `quantile()` function can be used to calculate sample quantiles.

Example 6.21 *The 15th and 75th percentiles of the distribution of V from Example 6.18 are*

```
Vq <- quantile(V, prob = c(.15, .75))
Vq

##       15%       75%
## 0.0185503 0.1674246
```

This means, for example, that the proportion of the 10000 simulated V's that are less than or equal to 0.167425 *must be at least 75% and the proportion that are less than* 0.167415 *is not more than 75%:*

```
mean(V <= Vq[2])

## [1] 0.75

mean(V <= Vq[2]-.00001)

## [1] 0.7499
```

The QQ-plot

A very effective way to compare the distribution of a sample with a theoretical distribution is to compare the sample quantiles with the corresponding theoretical quantiles. If they match or approximate each other well, we would have no reason to doubt the theoretical model as a reasonable approximation for the distribution that underlies the sample data.

A plot of the sample quantiles against the corresponding theoretical quantiles should consist of points that lie on or near a straight line through the origin with slope 1. Such a plot is called a Quantile-Quantile plot or QQ-plot.

Example 6.22 *A QQ-plot for the data simulated in Example 6.18 can be constructed using the following code:*

```
qU <- seq(.01, .99, .01)
qVsample <- quantile(V, prob = qU)
qVtheory <- 1/(1-qU)^(1/9) - 1
plot(qVsample ~ qVtheory, xlab = "Theoretical Quantiles",
    ylab = "Sample Quantiles")
abline(0,1)
```

The 1st through 99th percentiles of the V sample are computed in the first two lines, while, the third line computes the theoretical quantiles of the V distribution, using the fact that $V = g^{-1}(U) = 1/(1-U)^{1/9} - 1$. The graph is displayed in Figure 6.17 with a reference line, having 0 intercept and slope 1, overlaid.

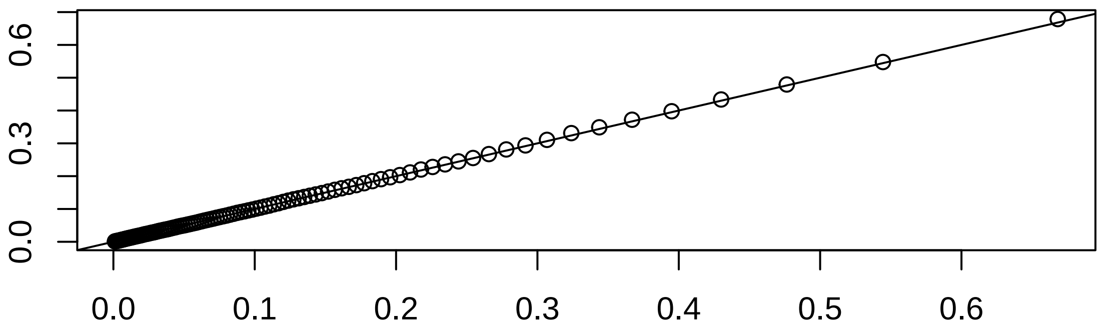


Figure 6.17: QQ-plot.

As expected, the points lie very near the reference line, indicating that the distribution of the simulated sample cannot be distinguished from the theoretical distribution.

6.2.6 Probability density functions

In general, for any continuous random variable V, we can express the cumulative distribution function $F_V(x)$ as an integral with upper endpoint at x. In many situations, we can find a function $f_V(x)$ for which we can write

$$F_V(x) = \int_{-\infty}^{x} f_V(y)dy.$$

The function $f_V(x)$ is called the probability density function of V, and is abbreviated as PDF.

Properties of density functions

Probability density is always nonnegative, and the area under the probability density curve is exactly 1.0. That is, the PDF of a random variable X satisfies the following properties:

$$f_X(x) \geq 0, \text{ for all } x$$

and

$$\int_{-\infty}^{\infty} f_X(x)dx = 1.$$

All probability density functions have these two properties.

Calculation of probabilities using the probability density function

The probability that a random variable X with density function $f_X(x)$ takes a value in an interval $[a, b]$ is calculated as

$$P(a \le X \le b) = \int_a^b f_X(x)dx.$$

Such probabilities are also expressed in terms of the *cumulative distribution function*:

$$F_X(y) = P(X \le y) = \int_{-\infty}^{y} f_X(x)dx.$$

Example 6.23 *Suppose model $f_X(x) = 0.5x^{-0.5}$ for $x \in [0, 1]$, and we seek the probability that X exceeds a given value v:*

$$P(X > v) = \int_v^{\infty} f(x)dx = \int_v^1 0.5x^{-0.5}dx = 1 - v^{0.5}.$$

Note that we are assuming $v \in (0, 1)$ here. If $v \ge 1$, the probability would be 0.

Recovering a PDF from a CDF

Given a CDF $F_V(x)$, we can obtain the PDF $f_V(x)$ by differentiation, if the CDF has a derivative. That is,

$$f_V(x) = F_V'(x).$$

Example 6.24 *For an exponential random variable with CDF $F_X(x) = 1 - e^{-\lambda x}$, the PDF is*

$$f_X(x) = F_X'(x) = \lambda e^{-\lambda x}.$$

Pareto distributions

If V is a random variable with cumulative distribution function

$$F_V(x) = 1 - \left(\frac{1}{x+1}\right)^k, \quad x > 0$$

and 0, otherwise, for some positive constant k, then V follows a Pareto distribution. (Sometimes, $X = V + 1$ is, instead, called a Pareto random variable.)

Samples with distributions like $F_V(x)$ are non-negative, right-skewed, and they tend to have outliers. The outliers become more extreme as k decreases. These distributions are commonly used in Economics as simple models for income levels and wealth distributions, since there are often many individuals with little wealth and a few with enormous wealth.

Example 6.25 *Suppose V is a random variable with CDF $F_V(v) = 1 - 1/(v + 1)^2$ for $x > 0$. Simulate 10000 random variates from this distribution.*

The inverse of this CDF is found by solving

$$U = 1 - 1/(v + 1)^2$$

for v:

$$1 - U = 1/(v + 1)^2$$

so that

$$1/(1 - U) = (v + 1)^2$$

which implies

$$v = 1/(1-U)^{1/2} - 1$$

or

$$F_V^{-1}(U) = 1/(1-U)^{1/2} - 1.$$

Employing this in the inverse CDF method runs as follows:

```
N <- 10000; U <- runif(N)
V <- 1/(1-U)^(1/2) - 1
```

We use the following code to display the histogram of the simulated values from the CDF $F_V(x)$, as in Figure 6.18. The figure in the right panel is of the cumulative histogram of the simulated data, with the CDF overlaid.

```
hist(V); chist(V); curve((1-1/(x+1)^2), add=TRUE)
```

```
par(mfrow=c(1,2), mar=c(2, 3, .75, .5))
hist(V, main="", xlab=""); chist(V); curve((1-1/(x+1)^2), add=TRUE)
```

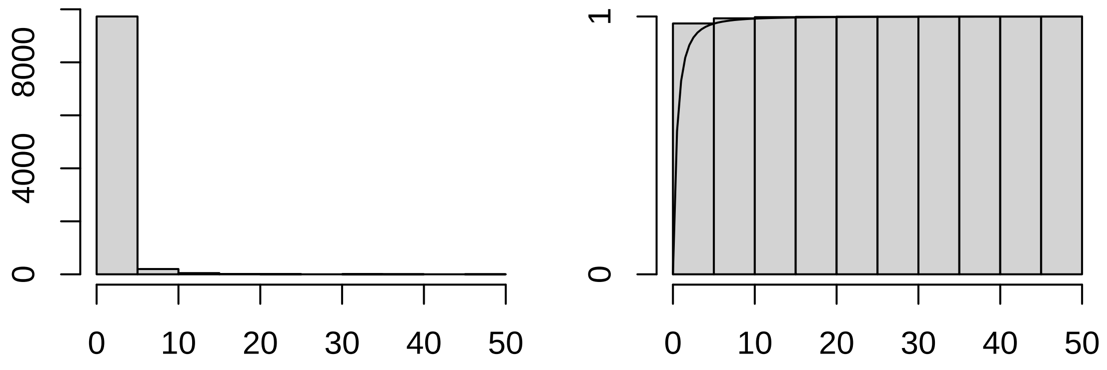


Figure 6.18: Histogram and cumulative histogram of 10000 simulated V values. See Example 6.25.

Again, the sample cumulative histogram and the true CDF match very well. Notice that the sample has a right-skewed distribution, and there are extreme outliers.

6.2.7 Expected value

The expectation operator $E[.]$ is a linear operator on random variables. The expectation of a single (continuous) random variable X can be written as

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx$$

where $f(x)$ is the probability density function of X. As for the uniform distribution, we say $E[X]$ is the *mean* of X. The expected value gives us a single number that, at least in a rough sense, conveys a typical value for the random variable. It is a measure of *location*, since it specifies the location of the distribution along the real axis.

Example 6.26 *For the probability density function $f_X(x) = (\alpha + 1)x^\alpha$, for $x \in [0, 1]$, we have*

$$E[X] = \int_0^1 x(\alpha + 1)x^\alpha dx = \frac{\alpha + 1}{\alpha + 2}. \tag{6.4}$$

For the specific case where $\alpha = -0.5$, this gives $E[X] = 1/3$.

A commonly used alternate notation for the mean of a distribution is μ, the Greek letter which roughly translates to the letter "m".

Example 6.27 *If X has an exponential distribution with parameter λ, then its mean μ is equal to $1/\lambda$:*

$$\mu = E[X] = \int_{-\infty}^{\infty} x f_X(x)dx = \int_0^{\infty} x\lambda e^{-\lambda x}dx = \frac{1}{\lambda}.$$

Integration by parts is needed to carry out the integration.

Note that an exponential distribution is often used as a model for the time to the occurrence of an event, so μ tells us the expected time to the occurrence, while λ is the rate of such occurrences.

Example 6.28 *Suppose V has PDF $f_V(x) = 2/(1+v)^3$, for $v \geq 0$, and 0, otherwise. Then*

$$E[V] = \int_0^{\infty} v\frac{2}{(1+v)^3}dv = 1.$$

To carry out the integration, you might try the variable substitution $u = 1 + v$.

Not all distributions have expected values.

Example 6.29 *If V has PDF $f_V(x) = 1/(1+v)^2$, for $v \geq 0$, and 0, otherwise, then*

$$E[V] = \int_0^{\infty} v\frac{1}{(1+v)^2}dv = \int_1^{\infty} \frac{1}{u}du + \int_1^{\infty} \frac{1}{u^2}du$$

where we have used the variable substitution $u = v + 1$. The first integral on the right is infinite, so the expected value of V is not finite.

The distribution in the preceding example is referred to as extremely heavy-tailed. This means that the probability density associated with large values does not decrease as quickly as lighter-tailed distributions, such as the exponential distribution. Thus, it is more prone to producing extremely large outlying values.

Other types of expected value can be calculated by the appropriate integration. For continuous functions $g(x)$, we have

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx.$$

Example 6.30 *Consider the distribution in Example 6.29, and the function $g(x) = 2x/(x+1)$. Then*

$$E[g(V)] = \int_0^{\infty} \frac{2v}{v+1}\frac{1}{(1+v)^2}dv = 1.$$

The integral becomes exactly the same as the one in Example 6.28.

When a is a nonrandom constant, and $g(x) = ax$, we have

$$E[aX] = \int_{-\infty}^{\infty} axf(x)dx = a\int_{-\infty}^{\infty} xf(x)dx = aE[X].$$

It can also be shown that

$$E[X + a] = E[X] + a.$$

Add something to a random variable, and the expected value of the variable will change by that amount.

Example 6.31 *If T is the boiling point of a liquid which is subject to random fluctuations in air pressure and with mean $E[T] = 100°C$, the expected boiling point of the temperature measurements if measured in Kelvin units is $E[T + 273] = E[T] + 273 = 373K$.*

Example 6.32 *Consider the PDF of Example 6.26 When $g(x) = x^2$, we have*

$$E[X^2] = \int_0^1 x^2(\alpha+1)x^\alpha dx = \frac{\alpha+1}{\alpha+3}.$$

Example 6.33 *If X is an exponential random variable with rate λ, then*

$$E[X^2] = \int_0^\infty x^2 \lambda e^{-\lambda x} = \frac{2}{\lambda^2}.$$

Two steps of integration by parts are needed to evaluate this integral.

6.2.8 Variance

We saw earlier that a feature of a distribution which is every bit as important as its location is its *scale*, or a measure of the degree of variability of the distribution. The variance (or its square root, the standard deviation) is one way to measure the variability of a random variable. Denoting the mean of X by μ, we have

$$\text{Var}(X) = E[(X-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx.$$

An algebraically equivalent expression is

$$\text{Var}(X) = E[X^2] - \mu^2.$$

Example 6.34 *For the PDF of Example 6.26, $f_X(x) = (\alpha+1)x^\alpha$, the variance is*

$$\textit{Var}(X) = \frac{\alpha+1}{\alpha+3} - \frac{(\alpha+1)^2}{(\alpha+2)^2} = \frac{\alpha+1}{(\alpha+3)(\alpha+2)^2}.$$

A small value of Var(X) implies that there is more certainty about the value of X; it will tend to take values close to μ when Var(X) is very small. The distribution will be more spread out when Var(X) is large.

The standard deviation is the square root of the variance. Like the variance, it summarizes the spread or variability in a probability distribution. It is sometimes preferred, since it is in the same scale as X, whereas the units for the variance are squared. Note also that

$$\text{Var}(aX) = a^2\text{Var}(X) \tag{6.5}$$

for any nonrandom constant a, and

$$\text{Var}(X+a) = \text{Var}(X). \tag{6.6}$$

In other words, the standard deviation of X is multiplied by a when X is. And the spread of the distribution doesn't change if it is only shifted by an amount a.

Example 6.35 *From Examples 6.27 and 6.33, we saw that an exponential random variable X has mean $E[X] = 1/\lambda$ and $E[X^2] = 2/\lambda^2$. Therefore, the variance of X is $1/\lambda^2$. The standard deviation is the square root of this: $1/\lambda$. Thus, the mean and standard deviation of an exponential distribution are the same.*

6.2.9 Calculating the mean and variance from a sample

When confronted with a sample of measurements $x_1, x_2, \ldots, x_n$, we can calculate the *sample mean* by taking the average of the sample values:

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j.$$

The *sample variance* is calculated as

$$s^2 = \frac{1}{n-1}\sum_{j=1}^{n}(x_j - \bar{x})^2.$$

The *sample standard deviation* is the square root of this: s.

Example 6.36 *We saw that the expected value and standard deviation of an exponential random variable are both $1/\lambda$. We will simulate $n = 1000$ variates from an exponential distribution with rate 3, and compute their sample mean and standard deviation with 1/3:*

```
x <-rexp(1000,rate=3)
mean(x); sd(x)

## [1] 0.3275
## [1] 0.3218
```

The average and standard deviation are close to each other and are also close to the theoretical value.

These functions can be applied to real data as well as simulated data.

Example 6.37 *For the sample of noon hour wind speeds contained in* `windWin80$h12`*, the sample mean, sample variance, and sample standard deviation are, respectively,*

```
mean(windWin80$h12)

## [1] 21.14

var(windWin80$h12)

## [1] 145.2

sd(windWin80$h12)

## [1] 12.05
```

This tells us that a typical wind speed at the Winnipeg airport is around 21 km/h, but the standard deviation of 12 km/h tells us that there is substantial variation in the individual speeds. A glance at the histogram displayed at the beginning of the chapter gives an even better impression of the variability of the windspeeds.

We can begin to see how to develop a reasonable model for the wind speeds by transforming the wind speeds themselves.

Example 6.38 *The sample mean and sample standard deviation are, respectively, after squaring:*

```
mean((windWin80$h12)^2)

## [1] 591.8

sd((windWin80$h12)^2)

## [1] 620.8
```

The similarity of the average and standard deviation are suggestive that the squared wind speeds follow an exponential distribution with a mean near 600.

The final example here hints at how we will start to apply our modelling and simulation tools to real data.

Exercises

1. Reconsider Example 6.17. Is $F_X(x)$ a true CDF? Show that the pdf is $f_X(x) = \cos(x)$ for $x \in [0, \pi 2/]$, and 0, otherwise.

2. Suppose U is a uniform random variable on the interval $[0, 1]$. Find $P(V \leq x)$ when V is defined as

(a) $\sqrt{U}$
(b) e^U
(c) $1/U$

3. A model for the time T (in hours) of failure of a flourescent light bulb is given by the Weibull distribution with shape parameter 5 and scale parameter 1000. Specifically, the probability that the light bulb would survive past time t is
$$P(T > t) = e^{-(t/1000)^5}.$$
Suppose U is a vector of uniform random numbers. Derive a formula which could be used to convert U into a vector of simulated failure times.

4. Suppose V has cdf $F_V(x) = 1 - \mathrm{e}^{-x^2}$ when $x > 0$, and is 0, otherwise.
   (a) Find the quantile function for V, and write an R function called `rmyV` which takes `n` as an argument and returns a vector containing `n` random variates from the distribution of V.
   (b) Simulate 10000 values from the distribution of V and
      i. display the values in a relative frequency histogram.
      ii. display the values in a cumulative frequency histogram, with the distribution function curve overlaid.
      iii. construct a QQ-plot of the sample quantiles with an appropriate reference line, for the purpose of checking that the numbers follow the distribution of V.
   (c) Determine the probability density function of V and plot the graph of the function. Compare with the histogram obtained in the previous question.

5. Suppose X has pdf $f_X(x) = x^2$, for $x \in [0, 1]$, and 0, otherwise.
   (a) Determine the cumulative distribution function of X.
   (b) Find the quantile function for X, and write an R function called `rmyX` which takes `n` as an argument and returns a vector containing `n` random variates from the distribution of X.
   (c) Simulate 10000 values from the distribution of X and
      i. construct a relative frequency histogram with the graph of the pdf overlaid.
      ii. display the values in a cumulative frequency histogram, with the distribution function curve overlaid.
      iii. construct a QQ-plot of the sample quantiles with an appropriate reference line, for the purpose of checking that the numbers follow the distribution of X.

6. Suppose p is a real number in the interval $(0, 1)$, and a random variable Y has pdf
$$g(y) = pf_V(y) + (1 - p)f_X(y)$$
where f_V and f_X are defined in questions 1 and 2.
   (a) Determine the cumulative distribution function of Y.
   (b) Write an R function called `rmyY` which takes `n` and `p` as arguments and returns a vector containing `n` random variates from the distribution of Y. (For this purpose, you will need to also use the `rbinom()` function, and the functions created in the previous exercises.)
   (c) Simulate 10000 values from the distribution of Y, for the case where $p = 0.4$. and
      i. construct a relative frequency histogram with the graph of the pdf overlaid.
      ii. display the values in a cumulative frequency histogram, with the distribution function curve overlaid.

7. Consider the pdf $h(x) = |x|\mathrm{e}^{-x^2}$, and suppose W has pdf $f_W(x) = ph(x - a) + (1 - p)h(x - b)$ for real constants a, b and $p \in (0, 1)$. Write a function called `rmyW` that takes arguments `a`, `b` and `n` and returns a vector of `n` random variates from the distribution of W. Obtain samples of 10000 Ws for the cases where $a = 1$ and $b = 3$, and where $a = 1$ and $b = 0.5$. Plot histograms with pdf curves overlaid.

6.3 Simulating from the family of normal models

6.3.1 Why does the normal distribution occur?

We encountered simulated normal random variates back in Example **??**, arising as a sum of independent uniform random numbers. Normality arises from summing many other kinds of random variables.

Example 6.39 *Consider a coin flipping game, where if one side appears, 1 unit will be paid out, and if the other side appears, there will be a cost of 1 unit. Suppose S_1 is the value of the first coin flip (± 1) and S_2 is the value of the second flip. Denote the average of the results as*

$$T = \frac{1}{2}(S_1 + S_2)$$

Possible values of T: $1, 0, -1$. The probability distribution of T is the following:

$$p(1) = .25, p(0) = .5, p(-1) = .25$$

Let's visualize the distribution of T, as in the upper left panel of Figure 6.19.

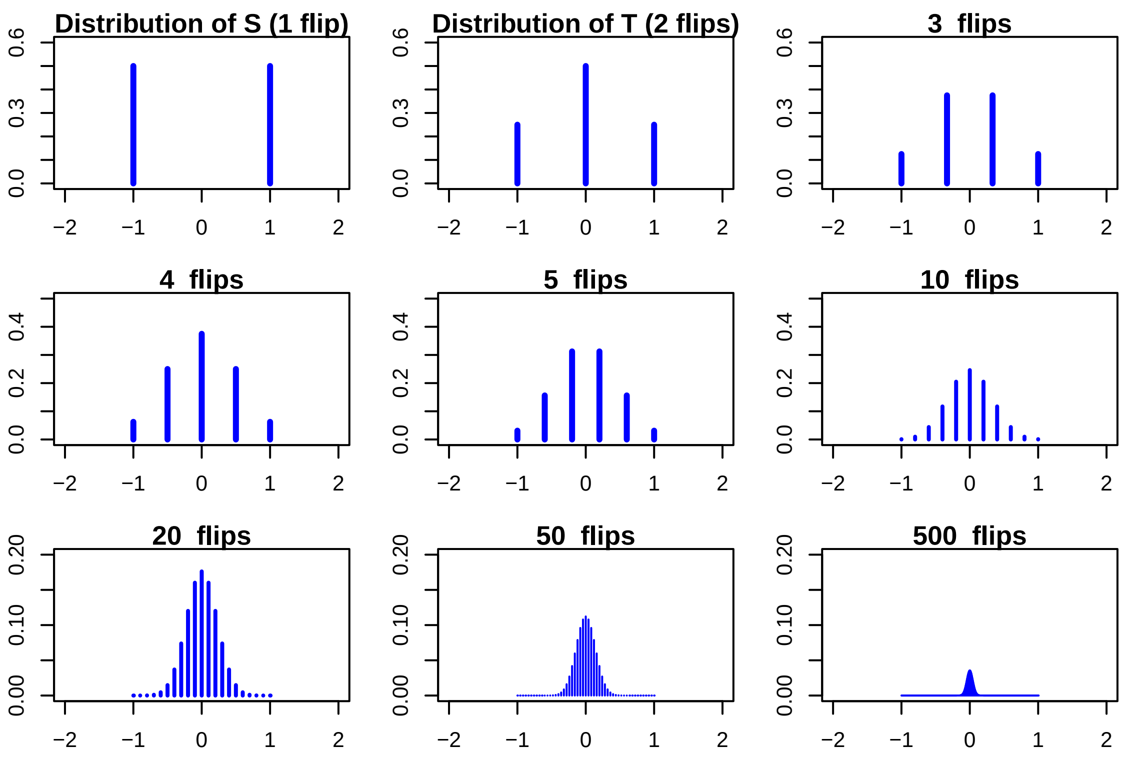


Figure 6.19: Distributions of coin flip game averages for various numbers of flips. Each flip counts as either $+1$ or -1.

The other panels of Figure 6.19 show what happens to the average of samples of equally likely 1's and -1's, for a growing sequence of sample sizes.

The previous example illustrates the concept behind the Central Limit Theorem. As the sample size increases, there are three observations that should be made:

1. The center of the distribution of the averages remains at 0, which is the expected value of one trial of the coin flip game.

2. The distribution adopts a bell-shape, characteristic of the normal distribution.
3. The spread of the distribution decreases, and the possible values become more dense within their overall range.

The following example shows that summing or averaging data from skewed distributions can also lead to a normal distribution.

Example 6.40 *We now simulate data coming from the distribution pictured in the top left panel of Figure 6.20. The distribution is skewed but has an expected value of 0.*

A large number of random samples of size $n = 1, 2, 5$, and 10 have been simulated, and the averages have been calculated and plotted as histograms. In each case, the sample mean and variance have been calculated as well.

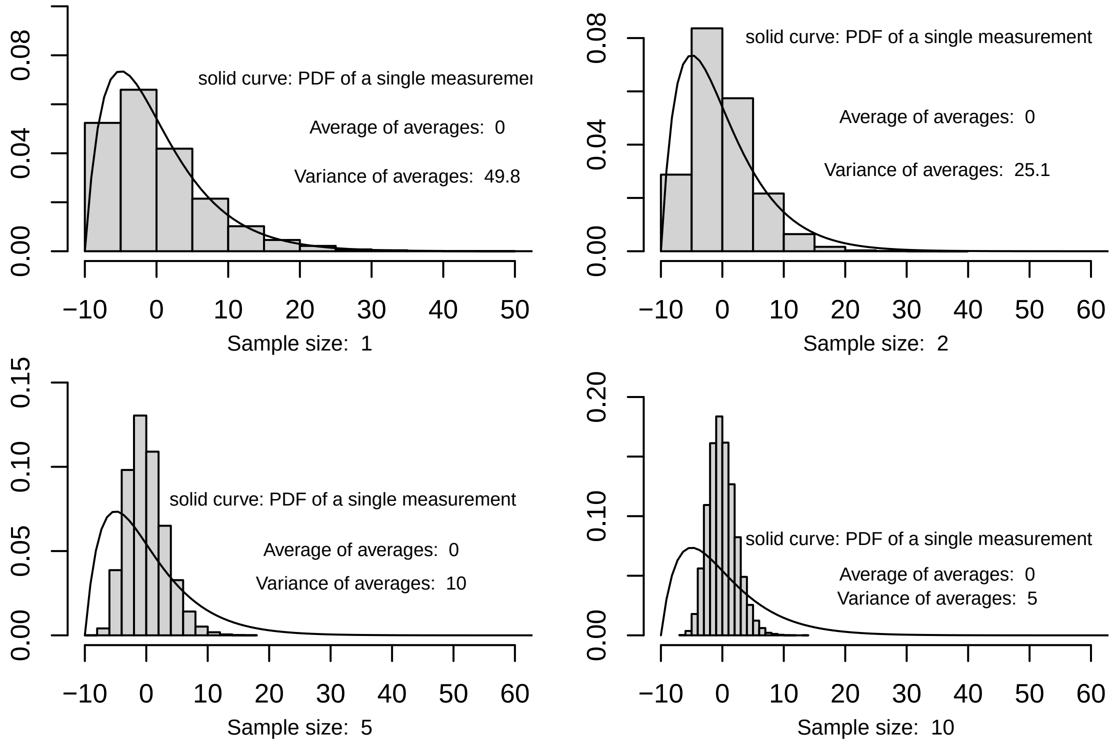


Figure 6.20: Histograms of averages of simulated samples of different sizes from a skewed distribution. The PDF for the distribution is overlaid on each histogram to emphasize that the data are originally coming from such a population.

Again, as the sample size increases (and in this case, we only obtained samples of size 10), the distributions of the sample averages appear to be more and more symmetric and bell-shaped. The overall averages are 0 and the variances decrease as the sample size increases. For samples of size 1, the variance is near 50, and the variance of averages of 2 measurements decreases by a factor of 2. With a sample of size 5, the variance decreases by a factor of 5, and the variance decreases by a factor of 10, when the sample size is 10.

The previous examples provide illustrations of the fact that the variance of averages of independent measurements is equal to the variance for a sample of size 1, divided by the sample size. If we take the average of n independent measurements X, each of which has variance σ^2:

$$\text{Var}(\bar{X}) = \sigma^2/n.$$

The examples also suggest that data might often appear to be normally distributed, or closely approximated by such a distribution, because data are often arrived at as the result of the addition of otherwise unmeasured quantities. For example, temperature is actually a measure of the average amount of heat in a large collection of molecules, the height of a human amounts to the sum of lengths of certain bones, and so on. Thus, it is unsurprising that human heights and temperatures seem to be normally distributed.

Central Limit Theorem

The examples in this section, together with Example **??** are special cases of the Central Limit Theorem. This theorem says that if $X_1, \ldots, X_n$ are independent and identically distributed random variables with mean μ and variance σ^2, then the distribution of

$$Z = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} \frac{X_j - \mu}{\sigma}$$

is approximately normal with mean 0 and variance σ^2/n, and the approximation improves as $n \to \infty$.

The independence property allows for cancellation of positive and negative parts of the added components; without this, averaging does not have the same effect.

Example 6.41 *As in Example 6.39, flip a coin once, getting Tails ($S = -1$), Let $T = (S + S)/2$. $T = -1$. If we keep on averaging the same value of S, we will keep on getting -1.*

This tells us that normality does not always arise from averaging. Dependence among the component elements is one consideration. The tendency for a distribution to have outliers is another as we will see next.

Averages of heavy-tailed data are not normal

It is also important that the variance of the distribution be finite. Recall that we simulated from Pareto distributions where the mean and variance did not exist. These heavy-tailed distributions do not obey the Central Limit Theorem, so averages of samples from these distributions will not be approximately normal and their variances will not decrease with sample size.

Example 6.42 *We will obtain 10000 samples of size 1, 10, 20 and 40 from a distribution that does not have an expected value or variance, and then calculate the variance of the averages of the samples.*

```
set.seed(196360) # to obtain the results below
N <- 10000; U <- runif(N)
V <- 1/(1-U)^(1/2) - 1
Vsum <- V
var(Vsum) # variance for samples of size 1

## [1] 6.023723

for (j in 2:10) {
    U <- runif(N)
    Vsum <- Vsum + 1/(1-U)^(1/2) - 1
}
var(Vsum/10) # variance for samples of size 10

## [1] 0.7479121

for (j in 11:20) {
    U <- runif(N)
    Vsum <- Vsum + 1/(1-U)^(1/2) - 1
}
var(Vsum/20) # variance for samples of size 20

## [1] 0.4457482
```

```
for (j in 21:40) {
    U <- runif(N)
    Vsum <- Vsum + 1/(1-U)^(1/2) - 1
}
var(Vsum/40) # variance for samples of size 40

## [1] 0.6498755
```

The first three variance values might lead one to the premature conclusion that the variance of the average really does decrease as the sample increases from 1 to 20, but then we see an increase in the variance from .446 to .650 when the sample increases from 20 to 40.

The previous example illustrates why conventional modelling techniques do not work for heavy-tailed or outlier-prone distributions.

6.3.2 Theoretical properties of the normal distribution

A standard normal random variable Z has a probability density function given by

$$f_Z(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}.$$

Using calculus, it is easily shown that if Z is a standard normal random variable, then

$$E[Z] = \int_{-\infty}^{\infty} z \frac{1}{\sqrt{2\pi}} e^{-z^2/2} dz = 0$$

and

$$E[Z^2] = \int_{-\infty}^{\infty} z^2 \frac{1}{\sqrt{2\pi}} e^{-z^2/2} dz = 1.$$

Therefore, the variance of the standard normal random variable is 1.

Nonstandard normal random variables

Suppose

$$X = Z\sigma + \mu.$$

We take this as our definition of a normal random variable with parameters μ and σ. The expected value is

$$E[X] = E[Z]\sigma + \mu = \mu$$

We can also see that

$$\text{Var}(X) = \sigma^2 \text{Var}(Z) = \sigma^2.$$

The standard deviation of X is σ.

A normal random variable X has a probability density function given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the expected value of X, and σ^2 denotes the variance of X. The *standard normal* random variable is thus a special case which has mean $\mu = 0$ and standard deviation $\sigma = 1$.

The normal density function can be evaluated using the `dnorm()` function. The density cannot be integrated in closed form; numerical methods are required, and these are implemented in the function `pnorm()`.

6.3.3 Simulation using the inverse CDF

The quantile function of the normal distribution can be obtained using `qnorm()`. Thus, we can simulate normal random variates using the inverse transform method. The following code could be used to simulate standard normal random variates.

```
U <- runif(n)
Z <- qnorm(U)
```

To simulate normal random variates with mean μ and standard deviation σ, we could add the further line

```
X <- mu + sigma*Z
```

This is implemented in the function `rnorm()` function where we could obtain the same results from

```
rnorm(n, mean = mu, sd = sigma)
```

6.3.4 Squaring standard normal random variables

If Z is standard normal, then Z^2 is obviously a nonnegative random variable. It is an example of a chi-squared random variable on 1 degree of freedom. Its expected value is 1.

If we sum a collection of independent chi-squared random variables, we obtain another chi-squared random variable where the degrees of freedom are obtained by summing the degrees of freedom of the original variables. The expected value of a such a chi-squared random variable is equal to its number of degrees of freedom. We denote the distribution of a chi-squared random on k degrees of freedom by the symbol $\chi^2_{(k)}$.

Chi-square random distributions play a useful role in understanding the nature of variability in data and in estimation uncertainty. Briefly, note that if Y is normally distributed with mean μ and standard deviation σ, then $Z = (Y - \mu)/\sigma$ is standard normal, and $(Y - \mu)^2/\sigma^2$ has a $\chi^2_{(1)}$ distribution, and if $Y_1, \ldots, Y_n$, then

$$\sum_{j=1}^{n} (Y_j - \mu)^2 \sigma^2$$

has a $\chi^2_{(n)}$ distribution. From above, the expectation of this last quantity is n, so a simple algebraic rearrangement tell us that

$$E[\sum_{j=1}^{n} (Y_j - \mu)^2 n] = \sigma^2.$$

That is, if we knew μ, then $\sum_{j=1}^{n} (Y_j - \mu)^2 n$ is an unbiased estimator for σ^2: the average of all possible estimates is the correct value of the parameter. If μ is unknown, and we replace it with the average of the Y's, we get the usual sample variance: The quantity $\sum_{j=1}^{n} (Y_j - \bar{Y})^2 \sigma^2$ turns out to have a $\chi^2_{(n-1)}$ distribution, so its expectation is $n - 1$, and the sample variance is an unbiased estimator for σ^2.

Simulating chi-squared random variates

One way of simulating chi-squared random variables on k degrees of freedom is then to simulate k independent standard normal random variables, square them and add them, as in

```
X <- numeric(n)
for (i in 1:k) {
    X <- X + rnorm(n)^2
}
X
```

The function `rchisq()` can be also used to simulate n chi-square random numbers on a given number of degrees of freedom. The following code accomplishes the same thing as the previous snippet.

```
X <- rchisq(n, df = k)
```

Figure 6.21 displays histograms of simulated chi-squared random numbers on 1, 2, 5, and 10 degrees of freedom. For small numbers of degrees of freedom the distribution is highly right-skewed, and as the number of degrees of freedom increases, the distribution becomes more symmetric. This is another manifestation of the Central Limit Theorem effect.

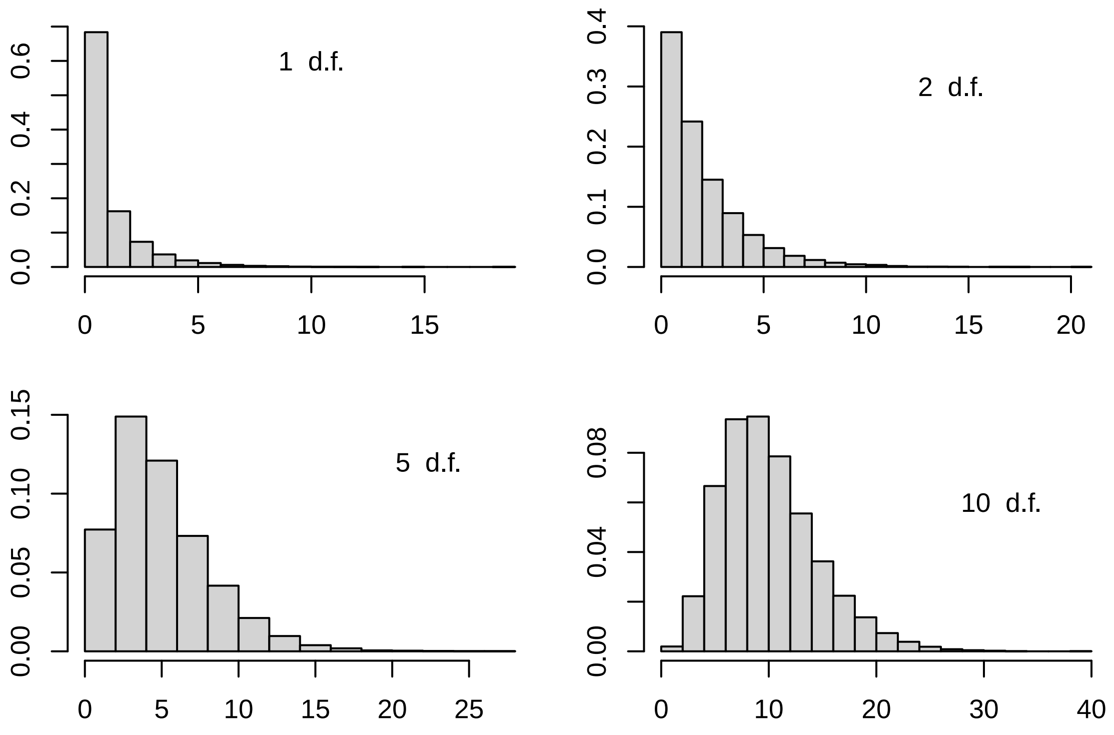


Figure 6.21: Histograms of simulated chi-squared random numbers on 1, 2, 5, and 10 degrees of freedom.

6.4 Simulating from models for survival times

The time from the present to an event of interests, such as a failure, death, or recovery is a nonnegative quantity, and is often associated with a tremendous amount of uncertainty. Thus, such times are often modelled as nonnegative random variables, and there are a large number of candidate distributions. We consider the most commonly encountered models in this section.

Recall the exponential distribution, a simple model for a lifetime distribution:

$$d_X(x) = \lambda e^{-\lambda x}, \quad x > 0$$

and $d_X(x) = 0$, otherwise. The graph of the PDF is plotted in Figure 6.22.

```
curve(dexp(x), 0, 5, ylab="f(x)")
```

The density is highest near 0. When we simulate from this distribution we get a lot of unrealistically low values, together with the occasional high value. For example,

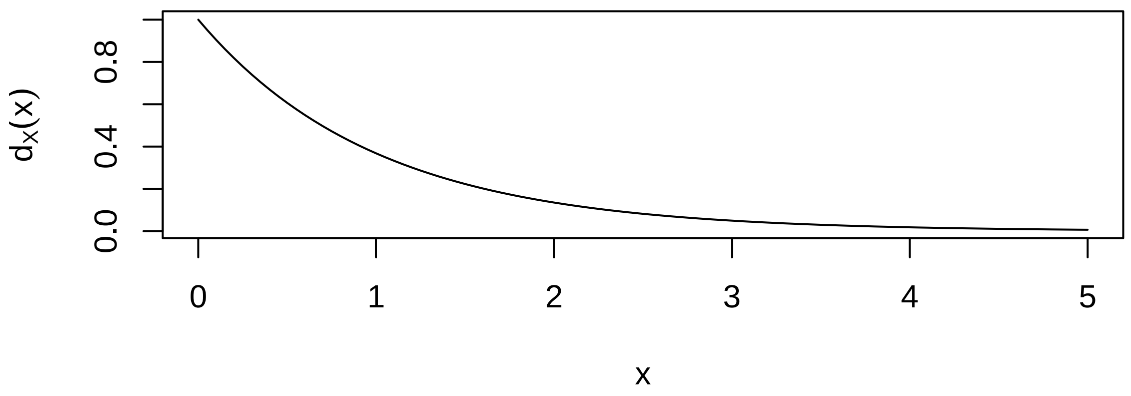


Figure 6.22: PDF of the exponential distribution with rate 1.

```
X <- rexp(9); X

## [1] 2.584 1.308 1.329 1.232 0.597 1.266 0.459 1.228 8.319
```

6.4.1 *The Weibull distribution*

If we take the square root of the exponentially distributed sample X, the behaviour becomes quite different:

```
sqrt(X)

## [1] 1.607 1.144 1.153 1.110 0.773 1.125 0.677 1.108 2.884
```

The small values have become larger and the large value has become considerably smaller. The right panel of Figure 6.23 shows the distribution of 10000 such Weibull variates, arising as the square roots of the exponential random variates giving rise to the histogram in the left panel. In the right panel, we see a distribution that has low density at small values and that is skewed to the right. Code to produce the plots is:

```
X <- rexp(10000)
Y <- sqrt(X)
par(mfrow=c(1, 2))
hist(X, main="", freq=FALSE); hist(Y, main="", freq=FALSE)
```

In general, a Weibull random variable is defined as a power of an exponential random variable. That is, if X is exponential, λ, then $Y = X^{1/\beta}$ is Weibull with parameters β and λ. β controls the shape of the distribution and λ controls the scale.

The CDF of Y is

$$F_Y(y) = P(Y \leq y) = P(X \leq y^{1/\beta}) = 1 - e^{-\lambda y^{1/\beta}}$$

where we used the exponential CDF of X in the middle of the above derivation. Differentiating $F_Y(y)$ yields the PDF of the Weibull distribution.

The PDF and CDF for the Weibull distribution can be computed using the following two functions:

```
dweibull(x, shape = 2, scale = 1) # Weibull PDF with beta = 2, lambda = 1
pweibull(x, shape = 2, scale = 1) # CDF
```

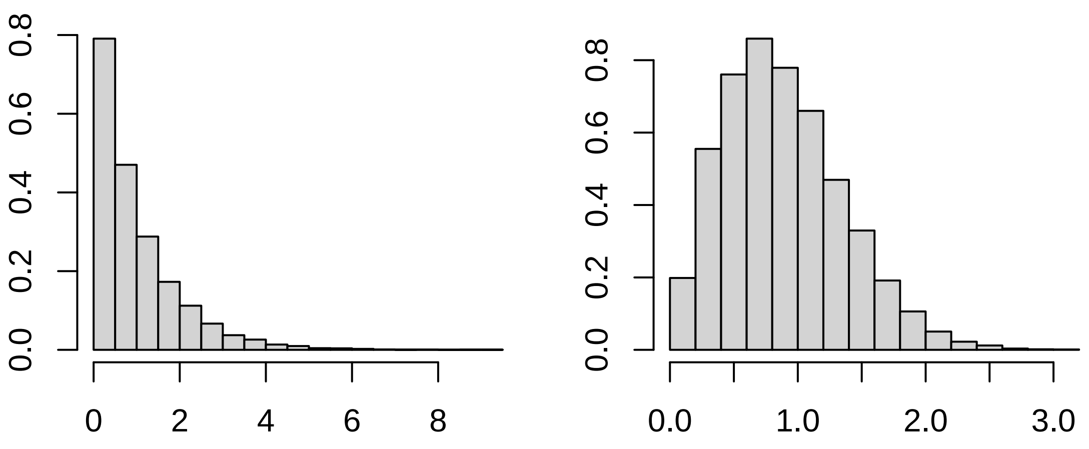


Figure 6.23: Histograms of exponential random variates (left panel) and their square roots (right panel) - Weibull variates with shape parameter 2.

Simulation of Weibull random variates with a built-in function

Since a Weibull random variable is a power of an exponential, it suffices to simulate the exponential and take the appropriate power. The following `rweibull()` function can also be used:

```
rweibull(n, shape = 2, scale = 1) # rng
```

Code to re-simulate Weibull data with shape parameter 2 with an overlaid density curve as in Figure 6.24 is as follows:

```
Y <- rweibull(10000, shape = 2, scale = 1)
hist(Y, freq = FALSE)
curve(dweibull(x, shape = 2, scale = 1), 0, 3, add = TRUE)
```

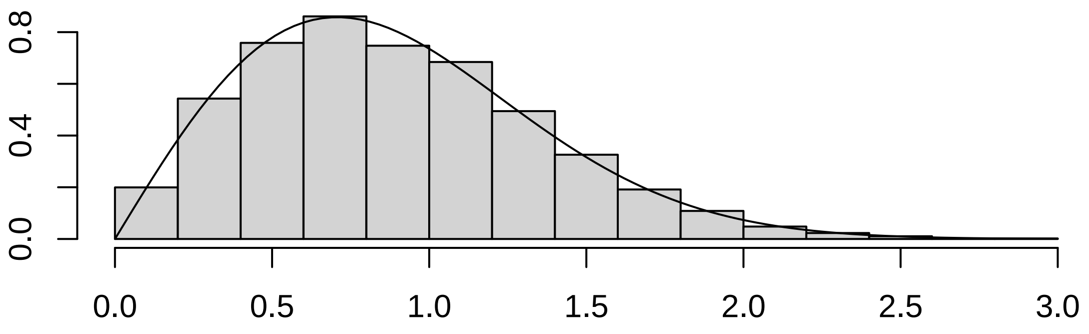


Figure 6.24: Histogram of simulated Weibull data with overlaid Weibull PDF.

6.4.2 The Lognormal Distribution

If we take the exponential of `X`, where `X` is a normal random variable, we obtain a lognormal random variable, another model for survival times. The following code does the transformation and plots a histogram as in Figure

6.25.

```
X <- rnorm(10000, mean = 1.5, sd = 0.5)
Y <- exp(X)
par(mfrow=c(1, 2))
hist(X, freq=FALSE); hist(Y, freq=FALSE)
```

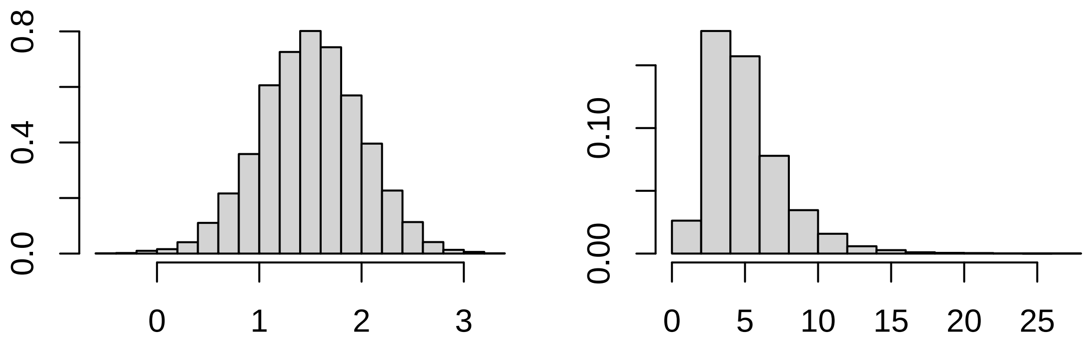


Figure 6.25: Histograms of simulated normal and corresponding lognormal data (right panel).

We now illustrate the effectiveness of a simple model like the lognormal distribution on some medical data.

Example 6.43 *Data in the* `transplant` *data frame in the survival package relate to waiting times for liver transplant patients. The* `abo` *column specifies whether the patient had type A, B or O blood. We consider the males who have type B blood here:*

```
library(survival) # this package contains the data
waitsMB <-  subset(transplant, sex=="m" & abo=="B" )$futime
```

These data are well approximated by the lognormal distribution, as can be seen in Figure 6.26. The raw data are in the left panel. The log of the raw data are in the right panel.

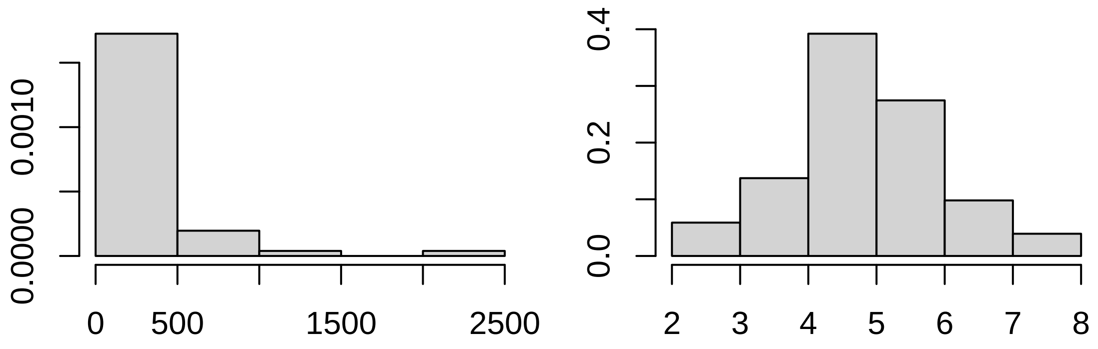


Figure 6.26: Histograms of raw liver transplant waiting time data (left panel) and logs of the data (right panel).

After taking logarithms, the distribution appears to be very close to normal. Figure 6.27 shows the normal QQ-plots on both the raw and log-transformed data, confirming the impression from the histograms: the log-transformed data are very close to normally distributed, meaning that the raw data are well approximated by a lognormal distribution.

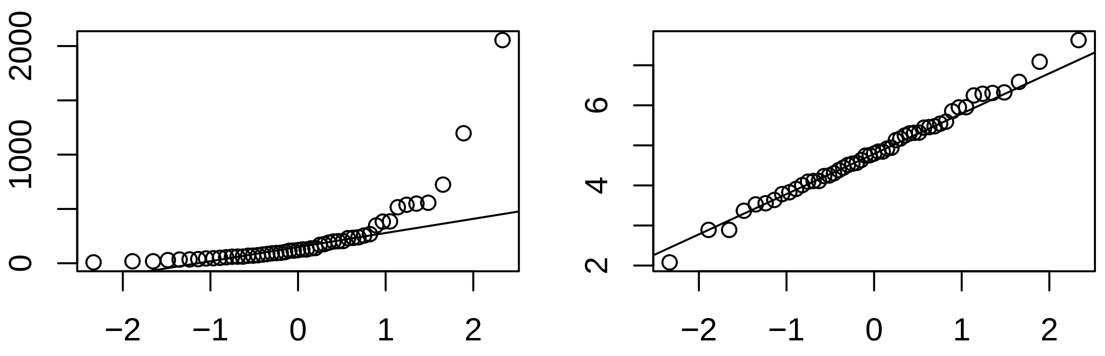


Figure 6.27: Normal QQ-plots of raw liver transplant waiting time data (left panel) and logs of the data (right panel).

Code to produce the figure is as follows:

```
par(mfrow=c(1, 2))
qqnorm(waitsMB, main=""); qqline(waitsMB)
qqnorm(log(waitsMB), main=""); qqline(log(waitsMB))
```

6.4.3 The gamma distribution

As discussed earlier, the PDF for the exponential distribution $d_X(x) = \lambda e^{-\lambda x}$ is limited in its usefulness because it only decreases as a function of x. When used as a model for the time to an event, the probability of a value near 0 is higher than might be reasonable.

As an alternative to raising an exponential random variable to a power, to obtain the Weibull distribution, premultiplying the exponential function by a power of x also gives a different shape to the distribution. The exponent of x is called the shape parameter and is usually denoted by the Greek letter α.

$$g_X(x) = kx^{\alpha-1}e^{-x} \quad \text{for } x \geq 0,$$

and 0, otherwise, where k is positive constant that depends on α. If $\alpha > 1$, then $g_X(0) = 0$, unlike the situation when $\alpha = 1$. This means that the tendency for very small values to occur is greatly diminished.

The gamma function

The integral of $g_X(x)$ over the nonnegative half-line is related to an object called the gamma function:

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}dx. \tag{6.7}$$

The symbol Γ is an upper case Greek letter which is called "Gamma". As suggested by the formula above, the gamma function depends on α. This function is only defined for positive values of α.

Among a large number of interesting properties that $\Gamma(\alpha)$ possesses, one that will be useful to us is the following:

$$\Gamma(\alpha) = (\alpha - 1)\Gamma(\alpha - 1), \quad \text{when } \alpha > 1. \tag{6.8}$$

This follows from application of one step of integration by parts to the integral in equation (6.7).

The gamma PDF

Dividing $x^{\alpha-1}e^{-x}$ by $\Gamma(\alpha)$ yields a function of x which integrates to 1. This function is the gamma PDF:

$$g_X(x) = \frac{x^{\alpha-1}e^{-x}}{\Gamma(\alpha)}.$$

Moments of the gamma distribution

Calculation of the expected value and variance of a gamma random variable is straightforward using the property at (6.8). We find that $E[X] = \alpha$, and $E[X^2] = \alpha(\alpha - 1)$. The variance can then be obtained as Var$(X) = \alpha$.

A more general gamma distribution

Recall that the exponential distribution was governed by a rate parameter λ. An exponential random variable X with rate 1, has expected value 1 and standard deviation 1. If we set $Y = X/\lambda$, then Y is still exponential but with mean $1/\lambda$ and standard deviation $1/\lambda$. This means that the scale of the exponential is related to $1/\lambda$, so we say that $1/\lambda$ is the scale parameter for the exponential distribution.

Similarly, we can define a scale parameter for the gamma distribution. If X is a gamma random variable with shape parameter α, and we set $Y = \beta X$, then we say that Y has a gamma distribution with shape parameter α and scale parameter β.

The gamma CDF and quantile function

The CDF for the gamma is not available in closed form, and neither is the quantile function. The functions `pgamma()` and `qgamma()` provide excellent numerical approximations.

Simulating gamma random variates

With the quantile function in hand, the inverse CDF method can be used to simulate random numbers from the gamma distribution.

Example 6.44 *Simulate 10000 random numbers from the gamma distribution with shape parameter 3 and scale parameter 1. Compare the mean and variance with their theoretical counterparts.*

```
n <- 10000; U <- runif(n)
X <- qgamma(U, shape = 3, scale = 1)
mean(X)

## [1] 2.99

var(X)

## [1] 2.99
```

What is the effect of multiplying the simulated values by 2.5 on the mean and variance?

```
Y <- 2.5*X
mean(Y)

## [1] 7.47

var(Y)

## [1] 18.7
```

The variable Y in this last calculation is a gamma random variable with shape 3 and scale 2.5. It is left as an exercise to determine what the theoretical mean and variance are for such a random variable.

In the previous example, we used the built-in quantile function to do the simulation. In fact, there is a built-in function that essentially performs the same operation: `rgamma()`. It takes `shape` and `scale` as input parameters.

Example 6.45 *Repeating the final simulation experiment of Example 6.44 using the built-in simulator, we have*

```
Y <- rgamma(n, shape = 3, scale = 2.5)
mean(Y); var(Y)

## [1] 7.46
## [1] 18.1
```

Special cases of the gamma distribution

Obviously, the exponential distribution is a special case of the gamma distribution, occurring when the shape parameter α takes on the value 1. The $\chi^2_{(k)}$ distribution is also a special case. In this case, the scale parameter β is 2, and $\alpha = k/2$. We note, in passing, that when $k = 2$, the chi-square distribution coincides with the exponential distribution. Thus, when Z_1 and Z_2 are independent standard normal random variables, $(Z_1^2 + Z_2^2)/2$ is an exponential distribution with rate 1, and it is not hard to argue further that $e^{-(Z_1^2+Z_2^2)/2}$ has a uniform distribution on $[0, 1]$ - which is precisely the ingredient needed to make the Box-Müller transformation work.

6.5 An application of simulation to integration

Often, integrals of the form

$$\int_a^b g(x)dx$$

are difficult or impossible to calculate in closed form. By thinking of this integral as an expected value with respect to a uniform distribution on the interval $[a, b]$, we can obtain a quick approximation to the integral by simulation: Monte Carlo integration.

To get started, suppose that U is a random variable on $[a, b]$, and write down the formula for the expected value of $g(U)$:

$$E[g(U)] = \int_a^b g(x)\frac{1}{b-a}dx$$

which is the integral we want to calculate divided by $b - a$. In other words, the integral we seek can be rewritten as $(b - a)E[U]$ where U is a uniform random variable on $[a, b]$.

Now, suppose $U_1, U_2, \ldots, U_n$ are independent uniform random variables on the interval $[a, b]$. We can calculate the sample mean of the U's as an estimate of $E[U]$, but what we really want is an estimate of $E[g(U)]$. The average of the transformed sample $g(U_1), g(U_2), \ldots, g(U_n)$: $\overline{g(U)}$ is such an estimator. Thus, the Monte Carlo estimate of the integral is $(b - a)\overline{g(U)}$.

Example 6.46 *Use Monte Carlo integration based on 50000 uniform pseudorandom numbers to estimate*

$$I = \int_1^4 x^2 dx.$$

```
n <- 50000
u <- runif(n, min=1, max=4)
integrand <- u^2 # this is g(U)
mean(integrand)*(4-1)

## [1] 21
```

Compare the output with the true value which is $(4^3 - 1)/3 = 21$.

Example 6.47 *Use Monte Carlo integration based on 150000 uniform pseudorandom numbers to estimate*

$$I = \int_0^2 \cos\left(x^2 \log(2x + 1)\right) dx.$$

```
n <- 150000
u <- runif(n, max=2)
integrand <- cos(u^2*log(2*u+1)) # this is g(U)
mean(integrand)*2

## [1] 0.716
```

In the first example, we were able to make a direct comparison of the Monte Carlo integral estimate with the true value, but in the second example, which is more realistic, we are unable to make such a comparison. An alternative way of assessing the error in the estimate is needed and discussed in the next section.

6.5.1 Estimating the error in the integral estimate

Since the estimate of the integral is now a multiple of a sample mean, it is possible to estimate the standard error of the estimate and to compute a confidence interval as described at the beginning of this book.

Recall that the standard error of the sample mean is the sample standard deviation divided by the square root of the sample size. Therefore, the standard error of $\overline{g(U)}$ can be estimated from the standard deviation of the sample $g(U_1), g(U_2), \ldots, g(U_n)$ divided by $\sqrt{n}$.

The standard deviation of $(b-a)\overline{g(U)}$ is obtained by multiplying the standard deviation of $\overline{g(U)}$ by $(b-a)$.

Example 6.48 *Returning to Example 6.47, we can estimate the standard error of the estimate of I:*

```
standev <- sd(integrand)
se <- 2*standev/sqrt(n)
se

## [1] 0.00369
```

6.5.2 Numerical integration

It is also possible to numerically integrate functions. A built-in function for this purpose is `integrate()`. To use the function successfully, three arguments are minimally needed: the function of x to be integrated, and the two limits of integration.

Example 6.49 *Use Monte Carlo integration to integrate the function $sin(e^{-x})$ on the interval $[0, 1]$. Compare with the result you would get using the* `integrate()` *function.*

The Monte Carlo integral:

```
N <- 1000000
X <- runif(N)
mean(sin(exp(X)))

## [1] 0.875
```

The numerical integral:

```
integrate(function(x) sin(exp(x)), 0, 1)

## 0.875 with absolute error < 9.7e-15
```

The values differ by .0003.

This example shows that numerical integration of a univariate function can be a lot more accurate than Monte Carlo integration.

6.5.3 Turning the standard error into a confidence interval

An approximate 95% confidence interval for the mean can be obtained by adding and subtracting 2 standard errors from the mean estimate.

Example 6.50 *Referring to the previous example, compute the standard error of the Monte Carlo estimate.*

```
sd(sin(exp(X)))/sqrt(N)

## [1] 0.000145
```

The value is around 0.00014, so this means that the Monte Carlo estimate should be within about .00028 of the true value (with 95% confidence).

6.6 Antithetic sampling

Recall the basic calculus result that says that

$$\int_0^1 \mathrm{e}^x dx = e^1 - 1 \doteq 1.71828.$$

A Monte Carlo estimate of this integral, using the simulated uniform variates from Example 6.49 can be calculated as

```
options(digits=7)
mean(exp(X))

## [1] 1.718448
```

However, note that if we perform the following calculation, using the same uniform variates, we obtain a much better result.

```
0.5*(mean(exp(X) + exp(1-X)))

## [1] 1.718301
```

This is an example of the use of antithetic sampling. We re-use the uniform variates, exploiting the fact that both X and $1-X$ are uniformly distributed on the interval $[0, 1]$, suggesting that both $E[\mathrm{e}^X]$ and $E[\mathrm{e}^{1-X}]$ are equal to $\int_0^1 \mathrm{e}^x dx$. (Use the substitution $y = 1 - x$ in this integral to see that it is equal to $\int_0^1 \mathrm{e}^{1-x} dx$). The average of the two unbiased estimators of the integral is also an unbiased estimator for the integral. In general,

$$\int_0^1 g(u)du = \int_0^1 g(1-u)du.$$

Thus, averaging of unbiased estimators not only preserves unbiasedness (i.e. that the estimators will be on target on average), but it is also almost guaranteed to reduce variance. This is based on the fact that the variance of a mean of two random variables Y and Z satisfies the relation

$$\text{Var}((Y+Z)/2) = \text{Var}(Y)/4 + \text{Var}(Z)/4 + \text{Cov}(Y, Z)/2.$$

If Y and Z have the same distribution, this simplifies to

$$\text{Var}((Y+Z)/2) = \text{Var}(Y)/2 + \text{Cov}(Y, Z)/2,$$

and the covariance is certain to be no more than the variance of Y. This means that, in general,

$$\text{Var}((Y+Z)/2) \leq \text{Var}(Y).$$

Thus, if Y and Z represent the estimators of the same integral, then the average of these estimators will have a variance that is no larger than the original estimator. Furthermore, since the covariance can be negative, the variance of the average can be substantially less than the variance of the original estimator.

For the example we began this section with, the covariance is negative, as seen by estimating the correlation:

```
cor(exp(X), exp(1-X))

## [1] -0.967643
```

In the case of Example 6.49, the covariance between the estimators is positive, but small:

```
cor(sin(exp(X)), sin(exp(1-X)))

## [1] 0.1615871
```

Therefore, we get an improvement in the estimate of the integral $\int_0^1 \sin(\mathrm{e}^x)dx$, but the improvement is not as dramatic as in the first example:

```
mean(sin(exp(X))) # original estimate

## [1] 0.8748816

0.5*(mean(sin(exp(X)) + sin(exp(1-X)))) # antithetic sampling estimate

## [1] 0.8749222
```

6.7 Rejection sampling

As we have seen, it is not always possible to come up with a convenient formula for the inverse of a CDF. Thus, other methods are often required to simulate random variables. Rejection sampling is one such method. The basic method is to *propose* a candidate value, simulated from another distribution, and then to perform a simple check to decide if the proposed candidate should be included in the sample or rejected. Obviously, the proposed value should be simulated from a distribution that is straightforward to sample from, such as the uniform distribution.

The rejection sampling concept, based on uniform proposals for the PDF

$$f_X(x) = (\alpha + 1)x^\alpha, \quad \text{for } x \in [0, 1]$$

is illustrated in Figure 6.28. The algorithm is as follows:

```
n <- 500
U <- runif(n)
alpha <- 1.5
fX <- function(x) (alpha + 1)*x^alpha
V <- runif(n)*(1+alpha)
accept <- V <= fX(U)
```

500 proposals are made from the uniform distribution on $[0, 1]$. Some of these will be accepted as variates from $f_X(x)$, and others will be rejected. To make the decision, values V are sampled from a uniform distribution whose range is larger than the maximum height of the target PDF (here, this value is $\alpha + 1$). The figure shows why accepting those values of U corresponding to V values that are less than $f_X(U)$ provides us with variates which follow the target PDF.

```
plot(V ~ U, xlim=c(-.5, 1.5), pch=1+15*accept, cex=.5)
curve(fX(x), -.5, 1.5, add=TRUE, lwd=2)
hist(U[accept], freq=FALSE, xlim=c(-.5, 1.5), main="")
curve(fX(x), -.5, 1.5, add=TRUE, lwd=2)
```

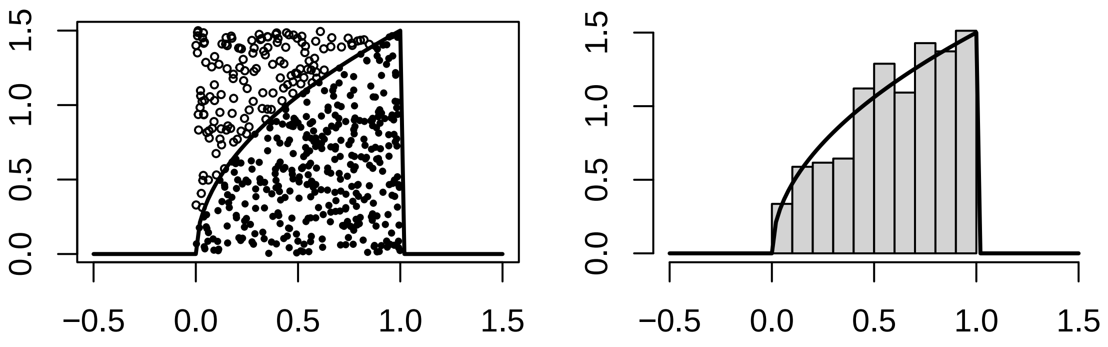


Figure 6.28: Illustration of rejection sampling concept. The solid black corresponds to the target PDF from which we wish to simulate random numbers. 500 pairs of uniform random variates are plotted in the left panel. The horizontal coordinates of those that are under the PDF curve in the left panel are used in the histogram in the right panel, which approximates the PDF. The accepted points (black dots) come from the target distribution.

In the above example, it is possible to obtain the CDF in closed form as well as the quantile function, so the inverse transform method could be used to simulate data from this distribution. The rejection sampling algorithm is most appropriate for use in situations where there is no closed form expression for the quantile function. It is also not necessary for the distribution to be restricted to values in the interval $[0, 1]$. We now consider such a situation.

Example 6.51 *Consider the biweight probability density function, plotted in Figure 6.29:*

$$f_X(x) = \begin{cases} \frac{15}{16}(1-x^2)^2, & \textit{for } x \in [-1,1] \\ 0, & \textit{otherwise} \end{cases}$$

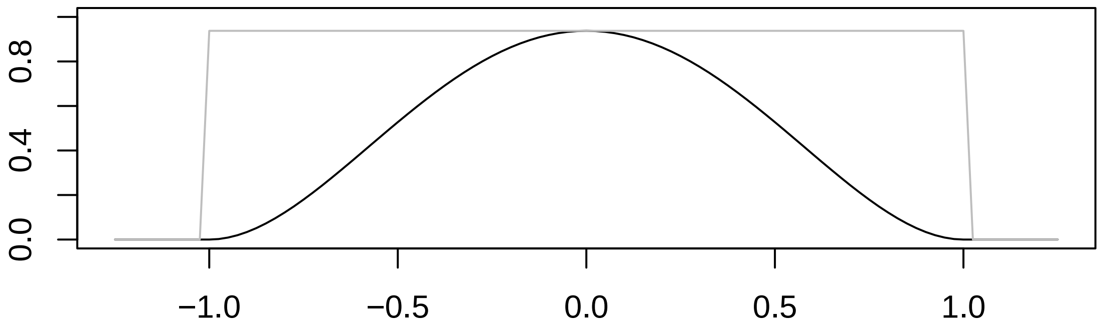


Figure 6.29: PDF of the biweight distribution.

Also included in the figure is a rectangle, outlined in grey, whose base matches the range of possible values in the biweight distribution and which has a height which is sufficient to match the maximum of the PDF.

Based on the figure, we see that we could sample from the biweight distribution by generating proposals U from the uniform distribution on the interval $[-1, 1]$. We can generate points in the grey rectangle by further simulating

V values from the uniform distribution on the interval $[0, M]$ where M is the maximum value of the PDF. This can be found using calculus, or by noting from the figure that the maximum occurs at $x = 0$ and so $M = 15/16$. The rejection algorithm then accepts those values of U for which $V \leq f_X(U)$, that is, those points in the rectangle that are below the $f_X(x)$ curve.

The following function implements this, taking n proposals from the $U[-1, 1]$ distribution, and returning those that are accepted:

```
rbiweight <- function(n) {
    U <- runif(n, -1, 1)
    V <- runif(n, 0, 15/16)
    X <- U[V < 15/16*(1-U^2)^2]
    return(X)
}
```

For example, if $n = 10$, we have

```
rbiweight(10)

## [1] -0.38204805  0.09253242  0.41653970  0.26146742  0.18059883
```

6.7.1 *Generating enough proposals to generate a full sample*

By its very nature, rejection sampling should rarely lead to a situation where all n proposals are accepted. Thus, more than n proposals will be needed to assure a sample of size n. An elegant way to generate exactly n random numbers is to use recursion. Here, the function containing the code for the generator calls itself, if too few proposals have been accepted. If n numbers are required, and the original rejection sample produces only m acceptances, the internal function call will be for $n - m$ new proposals. Of these, only a proportion will be accepted, so an additional function call will be required, and so on.

Recursion can be expensive in terms of storage, if many internal calls are made, so it is advisable to initiate the rejection method with more proposals than the required sample size. Because the rejection rate is roughly 30% in the biweight example, we can start with $1.5n$ proposals. This should ensure that only a few recursive calls will be required.

Example 6.52 *The following function simulates a random sample of* `n` *variates from the biweight distribution:*

```
rbiweight <- function(n) {
    u1 <- runif(1.5*n, -1, 1)
    u2 <- runif(1.5*n, 0, 15/16)
    x <- u1[u2 < 15/16*(1-u1^2)^2]
    if (length(x) < n) {
         return(c(x, rbiweight(n-length(x)))) # recursive call to rbiweight
    } else {
         return(x[1:n])
    }
}
```

We can use the above function to produce random samples of size 20, 100, 500 and 2000, and to construct a histogram in each case. The resulting plots are in Figure 6.30.

6.7.2 *The normalization constant is not required information*

The following function can be used to generate `n` pseudorandom numbers from the same distribution as above.

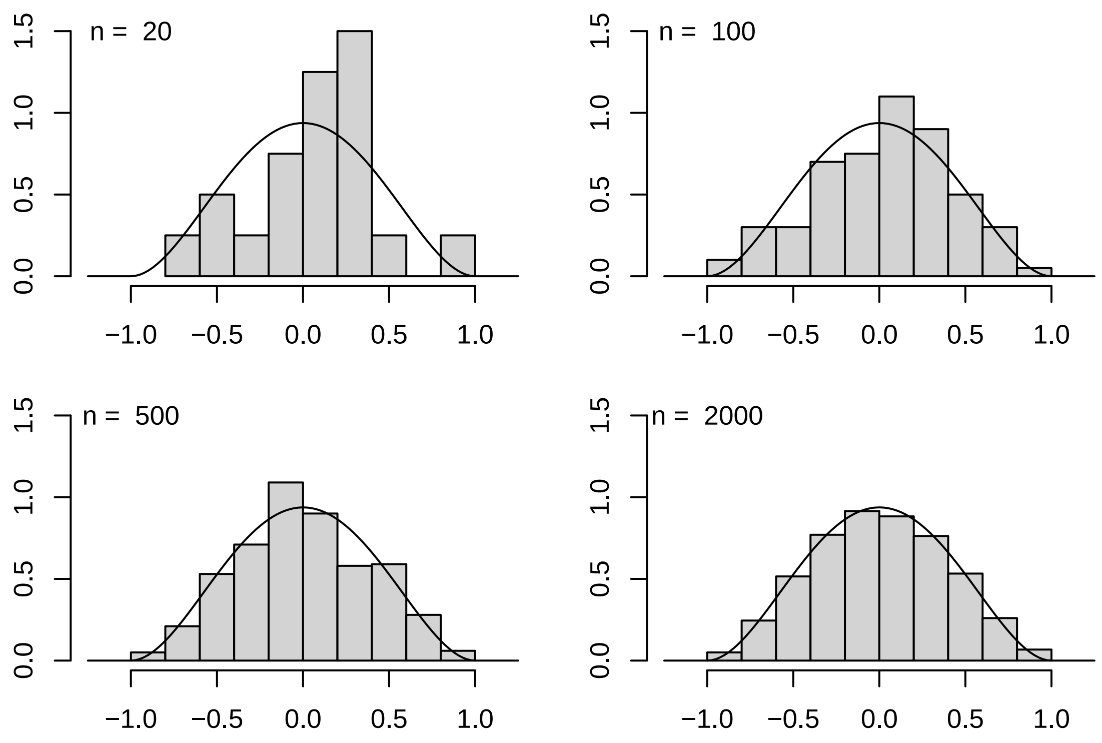


Figure 6.30: Histograms of samples from biweight distribution with samples of size 20, 50, 100, and 200, with overlaid PDF curve.

```r
rbiweight <- function(n) {
    u1 <- runif(1.5*n, -1, 1)
    u2 <- runif(1.5*n, 0, 1)
    x <- u1[u2 < (1-u1^2)^2]
    if (length(x) < n) {
        return(c(x, rbiweight(n-length(x))))
    } else {
        return(x[1:n])
    }
}
```

Note that it does not require knowledge of the constant $15/16$?. This is because the decision whether to accept or reject a proposed variate is based on the ratio of the target density height to the height of the bounding rectangle. The normalization constant had been in coded into both the numerator and denominator of this ratio, so it cancels out and is not needed. In fact, the constant can also be determined by Monte Carlo integration, and a confidence interval can be computed. This is done by inverting the lower and upper limits of the confidence interval for the integral.

Example 6.53 *For the biweight PDF, we seek the value of k for which*

$$\int_{-1}^{1} k(1-x^2)^2 dx = 1.$$

Thus, k is the multiplicative inverse of the integral $\int_{-1}^{1} k(1-x^2)^2 dx$. The Monte Carlo estimate of the integral is

```
n <- 100000
U <- runif(n, min = -1, max = 1)
BW <- (1-U^2)^2*(1--1) # the average of this estimates the integral
mean(BW)

## [1] 1.065498
```

The standard error of the integral is

```
sd(BW)/sqrt(n)

## [1] 0.002209704
```

Thus, a 95% confidence interval for the integral is $(1.061, 1.07)$. *Inverting these confidence limits gives the 95% confidence interval for* k*:*

$$(0.935, 0.942).$$

In this case, we can compare our result with the true value of $k = 15/16 = 0.938$ *which is comfortably inside the confidence interval.*

6.7.3 Simulating from the beta family of distributions

The beta distributions generalize the uniform distribution on $[0, 1]$, providing a variety of different shapes, depending on the two shape parameters, sometimes referred to as α and β. The PDF of a beta distribution is

$$f_B(x) = kx^{\alpha-1}(1-x)^{\beta-1}$$

where k is a constant that depends on α and β. The case of $\alpha = \beta = 1$ corresponds to the uniform distribution on $[0, 1]$. The function `dbeta()` evaluates the beta PDF. It takes arguments `x` as well as `shape1` (α) and `shape2` (β).

Figure 6.31 shows some of the different shapes that are obtainable from the beta model. The figure is based on the following code:

```
par(mfrow=c(2,2))
curve(dbeta(x, .5, 1.5))
curve(dbeta(x, 1.5, 1.5))
curve(dbeta(x, 3, 2))
curve(dbeta(x, 2, 3))
```

The CDF and quantile functions for the beta distribution can be evaluated with `pbeta()` and `qbeta()`. Random numbers can be simulated with the `rbeta()` function. The beta distributions with shape parameters that are at least 1 are natural candidates for rejection sampling.

6.7.4 Discontinuous probability density functions

The rejection sampling method is easily applied in cases where the PDF has discontinuities as illustrated in the next example.

Example 6.54 *This example is in the form of a worked set of exercises to give the reader practice in developing code for rejection sampling.*

Consider the cumulative distribution function

$$F_X(x) = P(X \le x) = \begin{cases} 0 & x \le 0 \\ 0.5x^3 & 0 < x \le 1 \\ 0.5x & 1 < x \le 2 \\ 1 & x > 2 \end{cases}$$

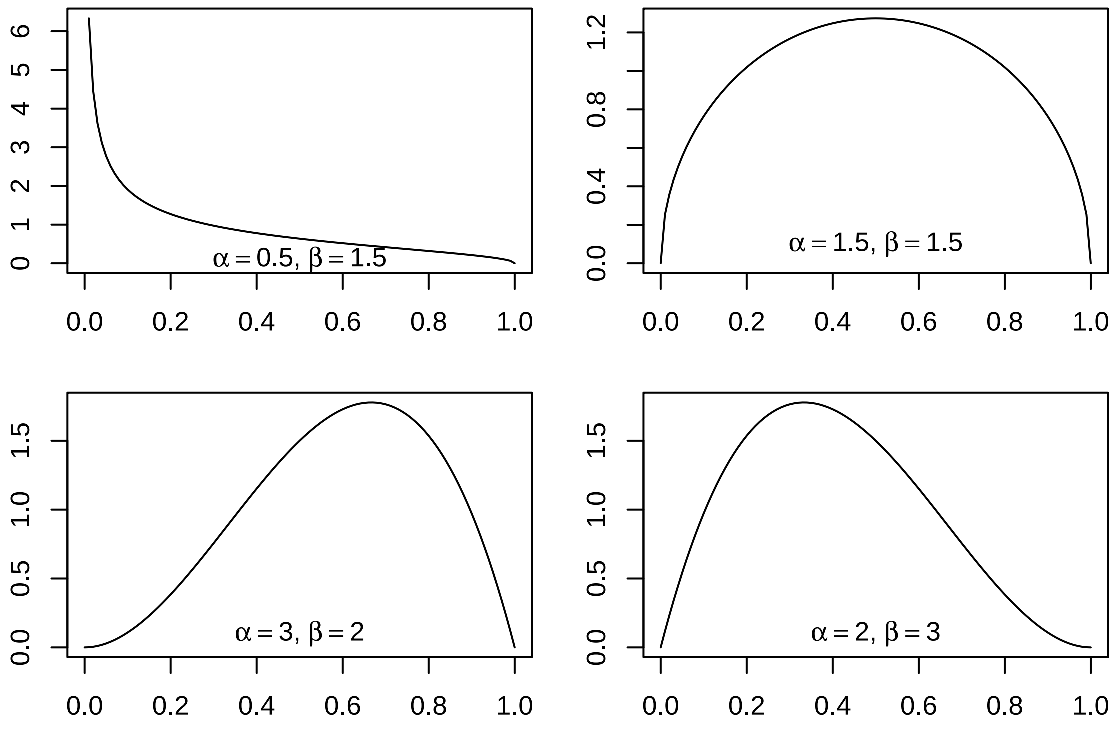


Figure 6.31: Various beta PDF curves.

1. *What is the probability density function?*

   The PDF is

$$f_X(x) = \begin{cases} 1.5x^2, & x \in (0,1] \\ 0.5, & x \in (1,2] \\ 0, & \textit{otherwise} \end{cases}$$

2. *Write an R function which evaluates the probability density function at a given vector* x. *Call the function* dfX.

```
dfX <- function(x) {
    1.5*x^2*(x > 0 & x <=1) + 0.5*(x > 1 & x <= 2)
}
```

3. *Write an R function which takes n as an argument and which returns a vector x containing n pseudorandom variates from the above distribution. (You will need to use the rejection sampling method, together with recursion. In order for the method to work, generate about $3n$ proposals at a time.)*

   The maximum value of $f_X(x)$ is 1.5 and the range of x is $(0,2)$. These observations are used to determine the interval from which the proposals are drawn, and the height of the rectangle bounding the target PDF.

```
rfX <- function(n) {
    u1 <- runif(3*n, max=2) # generated proposals
    u2 <- runif(3*n, max=1.5)
    u <- u1[u2 < dfX(u1)]   # accepted proposals
    if (length(u) > n) {
        return(u[1:n])
    } else {
        u <- c(u, rfX(n-length(u)))
        return(u)
    }
}
```

4. *Use the function just created to generate 100000 observations from the distribution $F_X(x)$. Assign these values to a vector called* `FXsample`*. Use* `set.seed(12345)` *beforehand.*

```
set.seed(12345)
FXsample <- rfX(100000)
```

5. *Use the result of the previous question to estimate $P(X > 1.5)$ when X has distribution function $F(x)$.*

```
mean(FXsample > 1.5)

## [1] 0.25
```

6.7.5 Using non-uniform proposals

More efficient use of proposals can be realized if the proposal density looks more like the target density. The next example shows that the proposal density does not have to be uniform.

Example 6.55 *Figure 6.32 shows how we can generate proposals from the beta(2,2) distribution to simulate variates from the triangular distribution on $[0, 1]$. The PDF for the triangular distribution is*

$$f_X(x) = \begin{cases} 2 - 2|1 - 2x|, & 0 \le x < 1 \\ 0, & \textit{otherwise} \end{cases}$$

This function takes values that are less than 4/3 times the corresponding values of the beta(2,2) PDF, as shown in the left panel of the figure with the grey and black curves. The proposals are shown to be under the grey curve, accepted (solid black points) when their values are less than the target PDF values.

Code to produce a large sample from the triangular density using the beta proposal density is:

```
U1 <- rbeta(100000, shape1=2, shape2=2)
U2 <- runif(100000)*dbeta(U1, shape1=2, shape2=2)*4/3
X <- U1[U2 < (2 - 2*abs(1 - 2*U1))]
```

Code to produce the histogram and PDF curve in Figure 6.33 is below. On this figure, we see that the technique has succeeded in producing variates from the target distribution.

```
hist(X, freq=FALSE)
curve(2*(1-abs(1-2*x)), -.5, 1.5, add=TRUE)
```

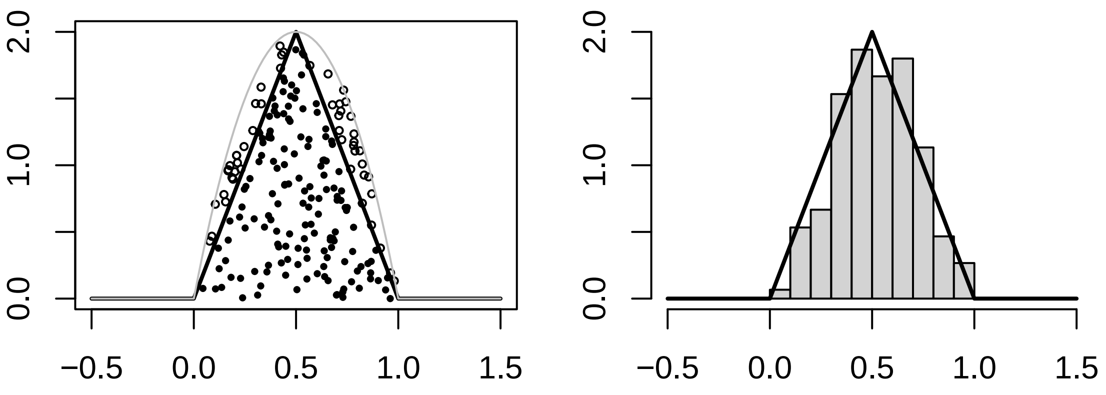


Figure 6.32: Illustration of rejection sampling concept. The solid black corresponds to the target PDF from which we wish to simulate random numbers. 200 pairs of beta random variates are plotted in the left panel. The proposal density (multiplied by 4/3) is plotted in grey. The horizontal coordinates of those that are under the target PDF curve in the left panel are used in the histogram in the right panel, which approximates the PDF. The accepted points (black dots) come from the target distribution.

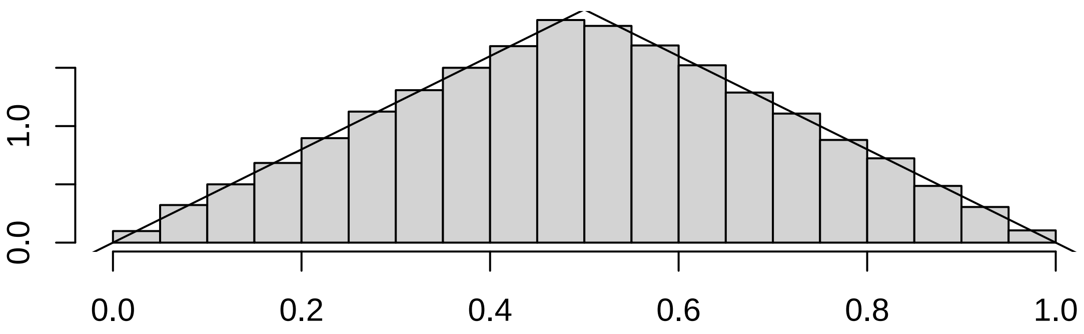


Figure 6.33: Histogram of random variates from the triangular density generated by rejection sampling using a beta proposal density.

6.7.6 General purpose rejection sampling

Rejection sampling is appropriate, not only for distributions with a finite range, but also those with an infinite range. In order for the procedure to work in such cases, it is necessary to find a proposal distribution which can be simulated from and for which the target PDF is bounded everywhere by (a multiple of) the PDF of the proposal distribution. That is, if $f_T(x)$ is the PDF of target distribution, and $f_P(x)$ is the PDF of the proposal distribution, the rejection sampling technique requires that $f_T(x) \leq cf_P(x)$ for all x, for some positive constant c.

Example 6.56 *Suppose we would like to simulate random variates from a target distribution having PDF*

$$f_T(x) = \frac{k}{x+1}x^{\alpha-1}e^{-x}$$

where k is an unknown normalization constant. For this problem, we can simulate from a gamma distribution having shape parameter α, because

$$f_T(x) \leq k\Gamma(\alpha)f_P(x)$$

where $f_P(x)$ is the PDF for a gamma random variable with shape parameter α and scale parameter $\beta = 1$. In this example, the k's cancel out, so we can take $c = \Gamma(\alpha)$.

The code to produce Figure 6.34 is below, for shape parameter $\alpha = 3$. The code proposes 100 variates U from the gamma distribution with $\alpha = 3$. Corresponding to each proposal is a variate V which is uniformly distributed on the interval $[0, cf_P(U)]$. The proposals are accepted if $V \leq fT(U)$. The left panel of the figure shows a scatterplot of all proposals (solid and open circles), demonstrating how the proposals are accepted, depending on whether they are below the target PDF curve.

```
n <- 100; alpha <- 3
U <- rgamma(n, shape = alpha, scale=1) # 100 proposals
V <- runif(n)*dgamma(U, shape=alpha, scale=1)*gamma(alpha)
k <- 1/integrate(function(x) x^(alpha-1)*exp(-x)/(x+1), 0, Inf)$value
fT <- function(x) x^(alpha - 1)*exp(-x)/((x+1))
X <- U[V <= fT(U)]
```

The right panel of Figure 6.34 shows the histogram of the accepted proposals and the overlaid target PDF. In order to plot the overlaid target PDF, we need to determine a value for the normalization constant k. Recall that this is a constant that ensures that the PDF integrates to 1. We can determine this value by Monte Carlo integration or by numerical integration of the function $\frac{1}{x+1}x^{\alpha-1}e^{-x}$ and inverting the result:

```
k <- 1/integrate(function(x) x^(alpha-1)*exp(-x)/(x+1), 0, Inf)$value
k

## [1] 1.68
```

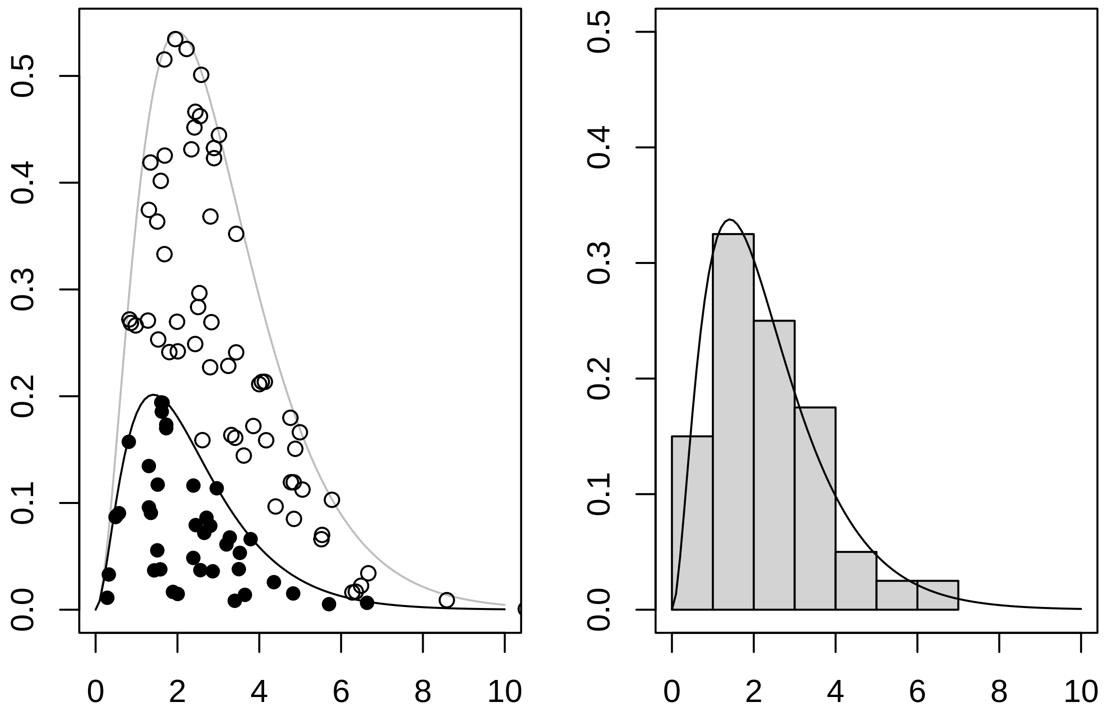


Figure 6.34: Left panel: 100 proposed values from gamma distribution (grey curve) with accepted values for unnormalized target PDF (black curve) of Example 6.56; Right panel: histogram of accepted values with overlaid normalized PDF curve.

Exercises

1. Show that the general gamma random variable with shape parameter α and scale parameter β has expected value $\alpha\beta$ and variance $\alpha\beta^2$.

2. Modify the `rbiweight` function so that it simulates `n` random values from the distribution with density

$$f(x) = \begin{cases} k(1 - \sin(\exp(x^2)))^2, & \text{for } x \in [-1, 1] \\ 0, & \text{otherwise} \end{cases}$$

   You will need to generate a much larger number of proposals, because the rejection rate will be in the order of 90% for this density. Plot a histogram of a simulated random sample of 100000 values from the above distribution.

3. Refer to the previous question.

   (a) Use Monte Carlo integration to estimate k.

   (b) Compute an approximate 95% confidence interval for k.

   (c) Use the point estimate of k, to overlay the density curve on your histogram.

4. Write a function that uses the rejection sampling method and simulates beta random variates. The function should take arguments `n`, `shape1` and `shape2` and returns a vector of length n. Does the function work if either of the shape parameters is less than 1?

5. Simulating normal random variates via rejection sampling is possible. The following code provides one example.

```
dfX <- function(x) {
    k <- 1/(1+2*exp(-1/2))
    d <- numeric(length(x))
    d[x >= 1] <- exp(-x[x >= 1]/2)*k
    d[x >= 0 & x < 1] <- k
d
}

n <- 100000
U <- runif(n)
X <- numeric(n)
k <- 1 + 2*exp(-1/2)
part1 <- U<(1/k)
X[part1] <- U[part1]*k
X[!part1] <- -2*log((1-U[!part1])*k/2)
V <- runif(n)*dfX(X)*sqrt(2/pi)*(1+2*exp(-1/2))
Z <- X[V <= 2*dnorm(X)]
B <- 1-2*rbinom(length(Z), size=1, prob=.5)
Z <- B*Z
```

   Investigate the code, and use appropriate graphics, to determine the shape of the proposal distribution and the shape of the target distribution. Check that the accepted proposals have been determined using the rejection sampling technique.

6. Use antithetic sampling to calculate the Monte Carlo estimate of $\int_0^1 \sin(\log(x+1))dx$. Compare with the original Monte Carlo estimate and with the result obtained with the `integrate()` function.

7. Use an appropriate substitution in the integral $\int_0^5 \sin(e^x)dx$ to obtain an integral of the form $\int_0^1 g(u)du$. Then apply antithetic sampling to obtain an estimate of this integral.

7

Modelling more than one continuous random variable

Until now, our focus has mostly been on the behaviour of a single random variable, using simulation and modelling tools to understand the visualize and describe univariate distributions. Now, we consider multivariate distributions, where several random variables are possibly interacting with each other. Our principle interest is in finding models for some of the patterns that relate two or more variables to each other.

7.1 Joint probability distributions

We focus on 2 measurements initially. Extensions to higher dimensions are relatively straightforward, but our treatment, later on, will be less detailed. For 2 observations, the probability density function (PDF) should be a function of 2 variables:

$$f_{X_1,X_2}(x_1,x_2).$$

In order for valid probabilities to be computed, it is necessary that this joint PDF be nonnegative, and its total (double) integral be 1.

Example 7.1 *Suppose x_1 and x_2 are random variables governed by the joint PDF*

$$f_{X_1,X_2}(x_1,x_2) = \frac{\alpha_1\alpha_2 x_1^{\alpha_1-1}x_2^{\alpha_2-1}e^{-x_1^{\alpha_1}/\beta_2 - x_2^{\alpha_2}/(\beta_0+\beta_1 x_1)}}{\beta_2(\beta_0+\beta_1 x_1)}, \quad \textit{for } x_1 \geq 0, x_2 \geq 0,$$

and 0, otherwise. The parameters $\alpha_1, \alpha_2, \beta_0, \beta_1$, and β_2 are nonnegative. A contour plot of this function is displayed in Figure 7.1 for the parameters $\alpha_1 = 3/2, \alpha_2 = 5/3, \beta_0 = 108, \beta_1 = 5.3$ and $\beta_2 = 65$.

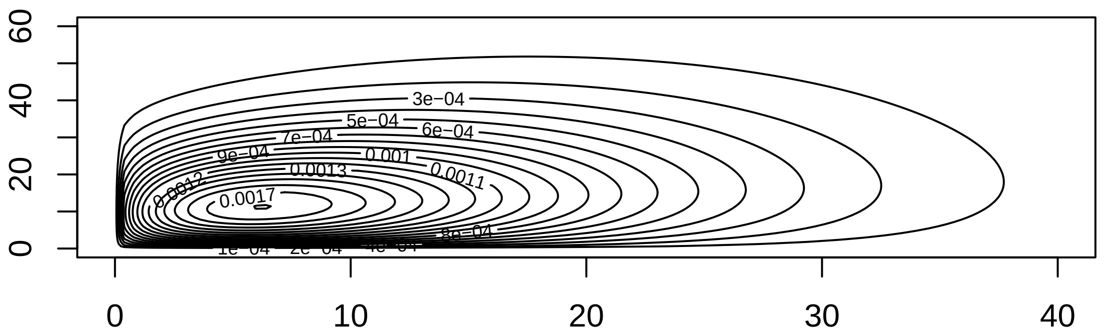


Figure 7.1: Contour plot of joint PDF described in Example 7.1.

Such a model provides a rough approximation to pairs of wind speed measurements. This one, in particular, matches data from the Winnipeg International airport where x_1 is the wind speed at midnight, and x_2 is the wind speed at noon on the following day. Observe that the PDF is maximum near $x_1 = 5$ and $x_2 = 10$, and the probability density decreases rapidly towards 0 as x_1 and x_2 decrease, and the PDF decreases less slowly as x_1

and x_2 increase. This means that it is possible, but not highly likely, that there would be a midnight wind speed of say 30 km/h followed by 40 km/h wind speeds the following noon.

Code to produce a 3D plot of the density:

```
wind2D <- function(x,y) a1*a2*exp(-x^a1/b2 - y^a2/(b0+b1*x))*x^(a1-1)*y^(a2-1)/(b2*(b0+b
)
library(rgl)
a1 <- 3/2; a2 <- 5/3; b0 <- 108; b1 <- 5.3; b2 <- 65
persp3d(wind2D, xlim=c(0,50), ylim=c(0, 60), polygon_offset=1, col="lightgreen", n=50, a
persp3d(wind2D, xlim=c(0,50), ylim=c(0, 60), front="lines", n=50, col="black", add=TRUE)
```

Example 7.2 *The proportion of impurity in iron ore samples is related to the quality of the resulting steel. Suppose the impurity proportion measurements coming from 2 iron ore samples have a joint PDF given by*

$$f_{X_1,X_2}(x_1,x_2) = (\alpha+1)^2(x_1x_2)^\alpha, \quad x_1,x_2 \in (0,1),$$

and 0, otherwise, where $\alpha > -1$. This PDF, with $\alpha = -0.25$ is displayed as a contour plot in Figure 7.2.

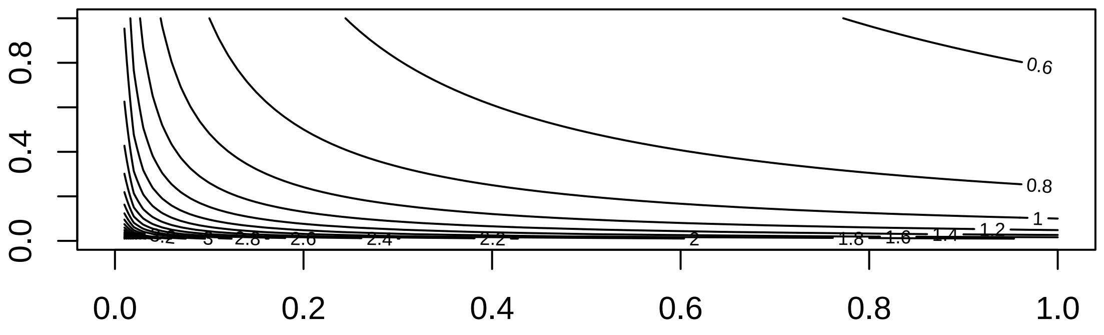


Figure 7.2: Contour plot of joint PDF described in Example 7.2.

In this case, the probability density decreases as x_1 and x_2 jointly increase.

7.1.1 Probability calculations

As for a single random variable, probabilities involving multiple variables are calculated by integration. The probability that X_1 lies in the interval (a,b) and X_2 lies in the interval (c,d) is

$$P(X_1 \in (a,b), X_2 \in (c,d)) = \int_a^b \int_c^d f_{X_1,X_2}(x_1,x_2)dx_1dx_2.$$

Example 7.3 *Returning to Example 7.2, we can evaluate probabilities concerning the proportion impurity in the two ore samples with the given joint PDF. Using calculus, it can be shown that*

$$\int_0^1 \int_0^1 (\alpha+1)^2(x_1x_2)^\alpha dx_1dx_2 = 1.$$

The probability that both impurity proportions exceed 0.75 is

$$\int_{0.75}^1 \int_{0.75}^1 (\alpha+1)^2(x_1x_2)^\alpha dx_1dx_2 = (1-0.75^{\alpha+1})^2.$$

Example 7.4 *A machine is used to automatically fill cylinders with propane gas. We suppose X denotes the amount of propane in a randomly selected cylinder (moles). and T denotes the temperature (in degrees Celsius) at the time of filling. A model for X and T could be based on the joint PDF:*

$$f_{X,T}(x,t) = \frac{x + \frac{t}{5} - 13}{5}, \; 10 \leq x \leq 11, \; 15 \leq t \leq 20$$

with $f_{X,T}(x,t) = 0$ for other values of x and t.

7.2 Expectation and covariance

Expected values are also calculated using double integrals:

$$E[g(X_1, X_2)] = \int \int g(x_1, x_2) f_{X_1,X_2}(x_1, x_2) dx_1 dx_2$$

where $g()$ is a continuous function of 2 variables, and the integrals are assumed to be taken over the support of the function $f_{X_1,X_2}(x_1, x_2)$ (i.e. wherever it is strictly positive). For the impurity proportion example,

$$E[X_1] = \int_0^1 \int_0^1 x_1(\alpha + 1)^2 (x_1 x_2)^\alpha dx_1 dx_2 = \frac{\alpha + 1}{\alpha + 2}.$$

An identical calculation shows that $E[X_2] = \frac{\alpha+1}{\alpha+2}$. Also,

$$E[X_1 X_2] = \int_0^1 \int_0^1 x_1 x_2 (\alpha + 1)^2 (x_1 x_2)^\alpha dx_1 dx_2 = \frac{(\alpha + 1)^2}{(\alpha + 2)^2}.$$

Example 7.5 *In Example 7.4, the pressure in the cylinder is proportional to XT. Suppose the relation is*

$$P = 3XT$$

Find $E[P]$.

$$E[P] = E[3XT] = \int_{10}^{11} \int_{15}^{20} 3xt \frac{x + \frac{t}{5} - 13}{5} dt dx$$

$$= \int_{10}^{11} \frac{105x^2 - 995x}{2} dx$$

$$= 568.75$$

The covariance of two random variables gives a measure of their association, and can be calculated using

$$\text{Cov}(X_1, X_2) = E[X_1 X_2] - E[X_1]E[X_2].$$

A nonzero value implies that the distributions of values of the pair of random variables will have (at least a vague) linear relationship. If the covariance is positive, high values of X_1 will tend to be associated with high values of X_2 and low values of X_1 will be associated with low values of X_2. When the covariance is negative, the opposite holds: high values of X_1 tend to occur with low values of X_2, and so on.

In the example considered earlier, we can show that $\text{Cov}(X_1, X_2) = 0$ which means that the two random variables do not have either a positive or negative linear association.

Dependent exponential random variables

For another example, consider random variables with the following joint probability density function

$$f_{X_1,X_2}(x_1,x_2) = \frac{\lambda}{x_1}e^{-\lambda x_1 - x_2/x_1}, \qquad x_1, x_2 \geq 0$$

and 0, otherwise. We can see that X_1 and X_2 are positively associated as follows.

$$E[X_1] = \lambda \int_0^\infty \int_0^\infty e^{-\lambda x_1 - x_2/x_1} dx_1 dx_2 = \lambda \int_0^\infty x_1 e^{-\lambda x_1} dx_1 = \frac{1}{\lambda}.$$

$$E[X_2] = \lambda \int_0^\infty \int_0^\infty \frac{x_2}{x_1} e^{-\lambda x_1 - x_2/x_1} dx_1 dx_2 = \frac{1}{\lambda},$$

and

$$E[X_1X_2] = \lambda \int_0^\infty \int_0^\infty \frac{x_1 x_2}{x_1} e^{-\lambda x_1 - x_2/x_1} dx_1 dx_2 = \frac{2}{\lambda^2}.$$

To compute these integrals, it is necessary to use integration-by-parts several times. Thus,

$$\text{Cov}(X_1, X_2) = \frac{2}{\lambda^2} - \frac{1}{\lambda^2} = \frac{1}{\lambda^2}.$$

This value is positive which means that X_1 and X_2 are positively related.

7.2.1 Correlation

The magnitude of the covariance, when nonzero, depends on the scale of the variables. For example, if the covariance between X_1 and X_2 is 1.0, it can be shown that the covariance between $10X_1$ and $10X_2$ is 100. By dividing through by the standard deviations of both variables, we can remove this dependence on scale. The resulting measure is the correlation:

$$\text{Corr}(X_1, X_2) = \frac{\text{Cov}(X_1, X_2)}{\sqrt{V(X_1)V(X_2)}}.$$

When the covariance is 0, the correlation will obviously be 0, so this is the case for the two impurity proportion measurements discussed earlier: we say these measurements are uncorrelated.

Positive covariances lead to correlations which are positive but no larger than 1. Negative covariances lead to correlations which are negative but no less than -1.

Calculation of covariance and correlation for a sample

For a sample $\{(x_{11}, x_{21}), (x_{12}, x_{22}), \ldots, (x_{1n}, x_{2n})\}$, the *sample covariance* is given by

$$c = \frac{1}{n-1} \sum_{j=1}^{n} (x_{1j} - \bar{x}_1)(x_{2j} - \bar{x}_2).$$

The *sample correlation* is given by

$$r = \frac{c}{s_1 s_2}$$

where s_1 and s_2 are the sample standard deviations of the samples of x_1's and x_2's respectively.

Example 7.6 *Brain and body weights for a sample of mice are recorded in the data set* `litters` *which can be found in the DAAG package Maindonald and Braun (2015).*

```
library(DAAG)    # load DAAG (after
  #  installing with install.packages("DAAG"))
litters
```

```
##    lsize bodywt brainwt
## 1      3  9.447   0.444
## 2      3  9.780   0.436
## 3      4  9.155   0.417
## 4      4  9.613   0.429
## 5      5  8.850   0.425
## 6      5  9.610   0.434
## 7      6  8.298   0.404
## 8      6  8.543   0.439
## 9      7  7.400   0.409
## 10     7  8.335   0.429
## 11     8  7.040   0.414
## 12     8  7.253   0.409
## 13     9  6.600   0.387
## 14     9  7.260   0.433
## 15    10  6.305   0.410
## 16    10  6.655   0.405
## 17    11  7.183   0.435
## 18    11  6.133   0.407
## 19    12  5.450   0.368
## 20    12  6.050   0.401

cor(litters$brainwt, litters$bodywt)

## [1] 0.7461485
```

Body weight and brain weight are positively correlated variables. A graphical interpretation of this statement is afforded by the plot in Figure 7.3 which is produced with the following code:

```
plot(brainwt ~ bodywt, data = litters)
```

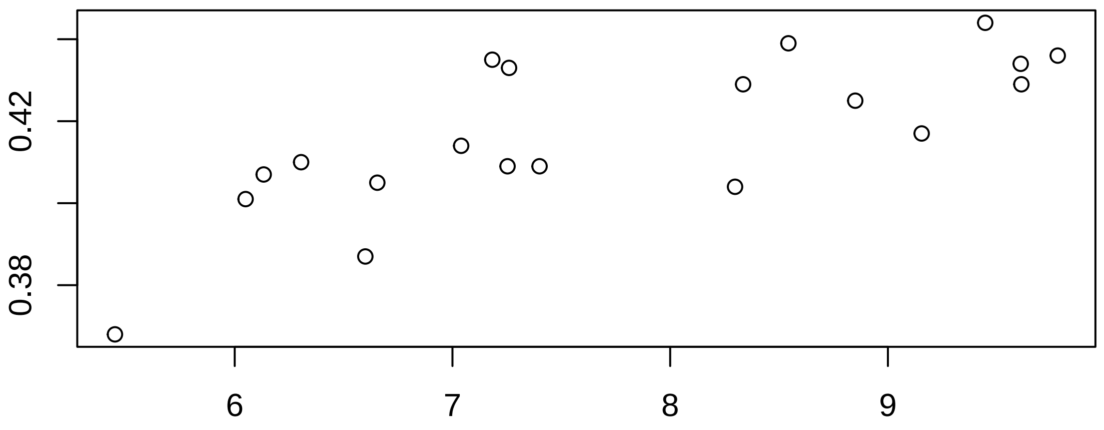


Figure 7.3: Graphical view of positively correlated data: the litters data of Example 7.6.

A best-fit line passing through these data would have a positive slope.

7.3 Marginal distributions

Earlier, we saw that

$$P(X_1 \in (a,b), X_2 \in (c,d) = \int_a^b \int_c^d f_{X_1,X_2}(x_1,x_2)dx_1dx_2.$$

What if we are only interested in $P(X_1 \in (a,b))$? In this case, it doesn't make a difference to us what the value of X_2 is; it could be any element of $(-\infty,\infty)$. In other words, we could write

$$P(X_1 \in (a,b)) = P(X_1 \in (a,b), X_2 \in (-\infty,\infty))$$

but, in integral form, this becomes

$$P(X_1 \in (a,b)) = \int_a^b \int_{-\infty}^{\infty} f_{X_1,X_2}(x_1,x_2)dx_2dx_1.$$

Recalling how we defined the probability density function for a single random variable, i.e.

$$P(X_1 \in (a,b)) = \int_a^b f_{X_1}(x)dx,$$

it necessarily follows that the probability density function for X_1 must be

$$f_{X_1}(x) = \int_{-\infty}^{\infty} f_{X_1,X_2}(x_1,x_2)dx_2.$$

Similar reasoning tells us that the probability density function for X_2 must be

$$f_{X_2}(x) = \int_{-\infty}^{\infty} f_{X_1,X_2}(x_1,x_2)dx_1.$$

To distinguish these probability density functions, if necessary, we will continue to write $f_{X_1}(x)$ as the probability density function of X_1 and $f_{X_2}(x)$ as the probability density function of X_2. $f_{X_1}(x)$ and $f_{X_2}(x)$ are referred to as marginal density functions.

To summarize: when in the context of several random variables, the marginal density of a single one of the random variables can be obtained by integrating the joint density function over all possible values of all of the random variables apart from the one of interest.

Example: proportion of impurity in iron ore samples

The marginal probability density function for X_1, the proportion of impurity observed in the first iron ore sample, is

$$f_{X_1}(x_1) = \int_0^1 (\alpha+1)^2(x_1x_2)^\alpha dx_2 = (\alpha+1)x_1^\alpha.$$

Similarly,

$$f_{X_2}(x_2) = (\alpha+1)x_2^\alpha.$$

Observe that the functional forms are the same for both X_1 and X_2. In this case, we say X_1 and X_2 are *identically* distributed.

Dependent exponential random variables

When the joint density function is

$$f_{X_1,X_2}(x_1,x_2) = \frac{\lambda}{x_1}e^{-\lambda x_1 - x_2/x_1}, \qquad x_1,x_2 \geq 0$$

we can obtain the marginal density function for X_1 by integrating over all possible values of x_2 (i.e. $x_2 > 0$):

$$f_{X_1}(x_1) = \int_0^\infty \frac{\lambda}{x_1}e^{-\lambda x_1 - x_2/x_1}dx_2 = \lambda e^{-\lambda x_1} \quad x_1 \geq 0$$

and 0, otherwise. We recognize this as the exponential density function.

Attempting to obtain the marginal density function for X_2 results in

$$f_{X_2}(x_2) = \int_0^\infty \frac{\lambda}{x_1} e^{-\lambda x_1 - x_2/x_1} dx_1$$

which can only be evaluated numerically. Clearly, X_2 does not have the same density function as X_1, so they are not identically distributed.

7.4 Conditional density functions

Suppose we know the value of X_1, and we would like to predict the value of X_2, using this information. The joint probability density function tells us how X_1 and X_2 are related, so we might think that $f_{X_1,X_2}(x_1, x_2)$ is the probability density function of X_2 for each given value of X_1, but this would be an incorrect interpretation. This is because

$$\int_{-\infty}^{\infty} f_{X_1,X_2}(x_1, x_2) dx_2 = f_{X_1}(x_1)$$

If $f_{X_1,X_2}(x_1, x_2)$ were a probability density function for X_2, given X_1, the above integral should evaluate to 1.

However, if we divide $f_{X_1,X_2}(x_1, x_2)$ by $f_{X_1}(x_1)$, we obtain a function of x_2 which integrates to 1:

$$\int_{-\infty}^{\infty} \frac{f_{X_1,X_2}(x_1, x_2)}{f_{X_1}(x_1)} dx_2 = \frac{f_{X_1}(x_1)}{f_{X_1}(x_1)} = 1.$$

It turns out that this gives a very useful predictive density function for X_2, given knowledge of X_1.

We write

$$f_{X_2|X_1}(x_1, x_2) = \frac{f_{X_1,X_2}(x_1, x_2)}{f_{X_1}(x_1)}$$

as the conditional density function of X_2 given X_1. This density function predicts the probability density of X_2, when we know the value of X_1.

Similar reasoning tells us that the predictive probability density of X_1 or its conditional density function is given by

$$f_{X_1|X_2}(x_1, x_2) = \frac{f_{X_1,X_2}(x_1, x_2)}{f_{X_2}(x_2)}$$

Example: impurity proportions

When

$$f_{X_1,X_2}(x_1, x_2) = (\alpha + 1)^2 (x_1 x_2)^\alpha, \quad x_1, x_2 \in (0, 1)$$

and

$$f_{X_1}(x_1) = (\alpha + 1) x_1^\alpha, \quad x_1 \in (0, 1)$$

it follows that

$$f_{X_2|X_1}(x_1, x_2) = (\alpha + 1) x_2^\alpha, \quad x_2 \in (0, 1).$$

Note that this conditional density function is the same as the marginal density of X_2, $f_{X_2}(x_2)$. Thus, X_1 does not give us any information about X_2. These random variables are independent.

Dependent exponential random variables

The situation is different when

$$f_{X_1,X_2}(x_1, x_2) = \frac{\lambda}{x_1} e^{-\lambda x_1 - x_2/x_1}, \qquad x_1, x_2 \geq 0$$

and

$$f_{X_1}(x_1) = \lambda e^{-\lambda x_1}, \quad x_1 \geq 0.$$

The conditional density for X_2, given X_1

$$f_{X_2|X_1}(x_1, x_2) = \frac{1}{x_1} e^{-x_2/x_1}, \quad x_2 \geq 0.$$

This is an exponential density function, but now the rate is $1/x_1$. This density function is not the same as the marginal density of X_2. Thus, X_1 gives predictive information about X_2. The two random variables are not independent.

Another way of extending univariate exponential distributions to the bivariate situation is to ensure that the conditional distribution of each of the variables, given the other is exponential.

Example 7.7 *Suppose $\theta > 1$, $k(\theta)$ is some unknown positive-valued function of θ, and*

$$f_{X,Y}(x, y) = k(\theta) e^{-(xy+\theta(x+y))}, \quad \textit{for } x \geq 0, y \geq 0$$

and 0, otherwise.

The function $f_{X,Y}(x, y)$ will be a proper probability density function as long as $k(\theta)$ is chosen to satisfy the condition that the double integral of $f_{X,Y}(x, y)$ over all nonnegative x and y is 1. Even though we cannot analytically determine $k(\theta)$, we can be certain that such a function exists as long as show that the everywhere positive function $e^{-(xy+\theta(x+y))}$ is bounded by a nonnegative function that integrates to a finite value. To see this requires a few steps. First, note that if $\alpha \geq 2$ and $\beta \geq 2$, then

$$\alpha\beta \geq \alpha + \beta$$

since $\alpha\beta > 2\beta$ and $\alpha\beta > 2\alpha$, and upon adding these inequalities, it follows that $2\alpha\beta > 2\alpha + 2\beta$. Next, note that, with the identification $\alpha = x + 2$ and $\beta = y + 2$, the previous result allows us to deduce the following:

$$xy + \theta(x + y) = (x + 2)(y + 2) + (\theta - 2)(x + y) - 4 \geq x + 2 + y + 2 + (\theta - 2)(x + y) - 4 = (\theta - 1)(x + y).$$

Therefore,

$$e^{-(xy+\theta(x+y))} \leq e^{-(\theta-1)x} e^{-(\theta-1)y}.$$

Integrating the right-hand side of this inquality over all $x \in [0, \infty)$ and $y \in [0, \infty)$ gives $1/(\theta - 1)^2$ which is a finite value for $\theta > 1$. This implies that the double integral of $e^{-(xy+\theta(x+y))}$ over the same quadrant must be some positive value that is even less than $1/(\theta - 1)^2$; by setting $k(\theta)$ to the reciprocal of this value, we can guarantee that the double integral of $f_{X,Y}(x, y)$ is 1.

We now proceed to find the conditional PDF of Y, given X. The first step is to determine the marginal PDF of X:

$$f_X(x) = k \int_0^\infty e^{-(xy+\theta(x+y))} dy = \frac{k}{x + \theta} e^{-\theta x}$$

where the integral in the middle of the above display is calculated in a very straightforward manner.

Dividing the joint CDF of X and Y by the marginal PDF of X gives us the conditional PDF of Y, given X:

$$f_{Y|X}(x, y) = (x + \theta) e^{-y(x+\theta)}.$$

Notice that the unknown $k(\theta)$ cancels out, so that we have all of the information we need about the conditional PDF of Y, given $X = x$. It is exponential with rate $x + \theta$.

We can go through analogous steps to find the conditional PDF of X, given $Y = y$. It is also exponential but with rate $y + \theta$.

7.5 Independence

We noted a case, earlier, where X_1 and X_2 are independent. In that case, X_1 provided no predictive information about X_2; the conditional density function of X_2 given X_1 was identical to the marginal density function of X_2:

$$f_{X_2|X_1}(x_1, x_2) = f_{X_2}(x_2).$$

Multiplying both sides of this by $f_{X_1}(x_1)$ gives the joint density function on the left and the product of the marginal density function on the right. Random variables are independent if their joint distribution can be factored in this way. That is, X_1 and X_2 are independent, if their joint density is

$$f_{X_1,X_2}(x_1, x_2) = f_{X_1}(x_1)f_{X_2}(x_2) \tag{7.1}$$

where the two functions are the respective marginal densities of X_1 and X_2.

7.6 Rejection sampling for bivariate distributions

In dimensions higher than 1, it is not usually possible to work out convenient expressions for quantiles. Therefore, alternative simulation methods to the inverse CDF transform method are needed. This is where the rejection method continues to be very useful. We illustrate the method with a single example here.

Example 7.8 *Rejection sampling can be used to simulate pairs of random variables from the PDF discussed in Example 7.7. The arguments in that example imply that*

$$f_{X,Y}(x, y) \le \frac{k}{(\theta - 1)^2} d_X(x)d_Y(y)$$

where $d_X(x)$ and $d_Y(y)$ are exponential PDFs with respective rates $\theta - 1$. Thus, we can bound the required PDF by a PDF formed as the product of two exponential PDFs: that is, we will simulate pairs of independent exponential random variables.

The fact that we do not know k is immaterial here, since it is the ratio of $f_{X,Y}(x, y)$ to $d_X(x)d_Y(y)$ that matters when accepting or rejecting a proposed (x, y) pair. More precisely, in order to accept a (x, y) pair, we require

$$\frac{f_{X,Y}(x, y)}{d_X(x)d_Y(y)} \le \frac{U}{(\theta - 1)^2}$$

where U is sampled from the uniform distribution on $[0, 1]$. The inline function `rejectStep()` in the code below implements this rejection procedure.

```r
rbexp <- function(n, theta = 2) {
    nremaining <- n
    rejectStep <- function(n, theta) {
        X <- rexp(n, rate = theta - 1); Y <- rexp(n, rate = theta - 1)
        V <- runif(n)*dexp(X, rate = theta - 1)*dexp(Y, rate = theta - 1)/(theta-1)^2
        ACCEPT <- V <= exp(-(X*Y + theta*(X+Y)))
        X <- X[ACCEPT]; Y <- Y[ACCEPT]
        cbind(X, Y)
    }
    XY <- cbind(c(), c())
    while (nremaining > 0) {
        XY <- rbind(XY, rejectStep(nremaining, theta = theta))
        nObs <- nrow(XY)
        nremaining <- n - nObs
    }
return(XY[1:n, ])
}
```

The `while()` loop is used, because only a fraction of the n proposed pairs are initially accepted. If the procedure stopped at that point, the user who had requested n pairs would be disappointed. Instead, the procedure is repeated until the full set of random pairs has been accepted.

Output from the function is in the form of an $n \times 2$ matrix. The following is a result when $n = 4$:

```
rbexp(4, 2)

##                X         Y
## [1,] 0.3095440 0.6437717
## [2,] 0.9720773 0.5182498
## [3,] 0.2359814 0.5165756
## [4,] 0.2813457 0.4091536
```

Checking the correctness of the simulation in higher dimensions than 1 is not straightforward. One, fairly awkward, way is to compare a contour plot of a two-dimensional density estimate (essentially a smoothed bivariate histogram) of the data with level sets (i.e. contours) of the joint PDF.

Example 7.9 *We can compute the contour plot of the simulated bivariate data from Example 7.8 as follows. First, we simulate a large sample.*

```
xy <- rbexp(1000000) # simulate a very large sample
out <- bkde2D(xy, bandwidth=.045) # compute the smoothed histogram
```

Next, we load the KernSmooth package which contains a two dimensional density estimator called `bkde2D`*. It requires a smoothing parameter, and we use a very small value - essentially like computing a histogram with very narrow bins. The contours of the density estimate are computed with the* `contour()` *function, which requires 3 arguments, representing the x, y and z coordinates of the object being drawn. Those coordinates come from the list output of* `bkde2D` *as* `x1`, `x2` *and* `fhat`.

The resulting contours are pictured as solid black curves in Figure 7.4.

```
library(KernSmooth) # load a package which
out <- bkde2D(xy, bandwidth=.045) # compute the smoothed histogram
 #  computes a bivariate smoothed histogram
plot(1, 1, xlim=c(0.1, 1.5), ylim=c(0.1, 1.5), type="n")
with(out, contour(x = x1, y = x2, z = fhat, add=TRUE))
curve((3-2*x)/(x+2), 0, 1.5, add=TRUE, lty=2, lwd=2) #overlay a theoretical contour
curve((2-2*x)/(x+2), 0, 1.5, add=TRUE, lty=2, lwd=2)
curve((1-2*x)/(x+2), 0, 1.5, add=TRUE, lty=2, lwd=2)
```

The theoretical contours, plotted as dashed curves in Figure 7.4, are obtained by determining the relation between x and y that holds when the PDF takes on a fixed value. That is, if we suppose that

$$e^{-xy-2(x+y)} = c$$

for some positive constant c, we take logs of both sides of this equation and perform some algebra that suggests that

$$y = \frac{-\log(c) - 2x}{x+2}.$$

The code above implements this for various values of $\log(c)$.

Alternative ways to visualize the simulated data in order to compare it with the theoretical distribution from which it is supposed to be sampled require information about the theoretical marginal and/or conditional distributions.

7.7 Gibbs Sampling

7.8 Multiple integration

We saw earlier that Monte Carlo integration can be much less accurate than numerical integration. This is generally true for single integrals, but numerical integration can often be less accurate for multiple integration, and the

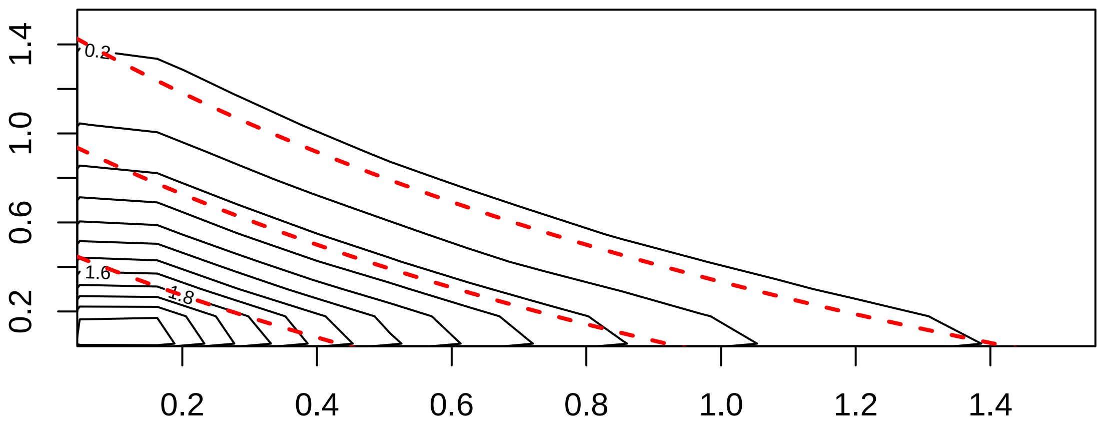


Figure 7.4: Contour plot of estimated bivariate PDF for data simulated from joint exponential model corresponding to Example 7.7 with some theoretical contours (dashed curves).

methods can be quite unwieldy, especially if the region to be integrated over is complicated. By contrast, multiple integration via Monte Carlo is a straightforward extension of the single integral method.

The Monte Carlo approach to integrating the double integral

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2} g(x_1, x_2)dx_1dx_2$$

goes as follows.

Let $V_1, V_2, \ldots, V_n$ be an additional set of independent uniform random variables on the interval $[0, 1]$, and suppose $g(x, y)$ is now an integrable function of the two variables x and y.

So we can approximate the integral $\int_0^1\int_0^1 g(x, y)dxdy$ by generating two sets of independent uniform pseudo-random variates, computing $g(U_i, V_i)$ for each one, and taking the average.

Example 7.10 *Approximate the integral $\int_0^4\int_3^5 e^{-xy}dxdy$ using the following:*

```
U <- runif(10000, min = 0, max = 4)
V <- runif(10000, min = 3, max = 5)
mean(exp(-U*V))*4*2

## [1] 0.5112305
```

Calculating the standard error of this estimate is left as an exercise.

7.9 Modelling sequences of random variables

When dealing with more than two random variables, the concepts of joint PDF, marginal PDF, conditional PDF and expected value continue to apply, but the dimensionality of the problem becomes larger, and the integrals are no longer double, but perhaps triple, quadruple or whatever the dimensionality requires. The joint PDF for a set of m random variables $X_1, X_2, \ldots, X_m$ has the form

$$f_{X_1,X_2,\ldots,X_m}(x_1, x_2, \ldots, x_m)$$

and has the properties that it is

1. nonnegative everywhere.

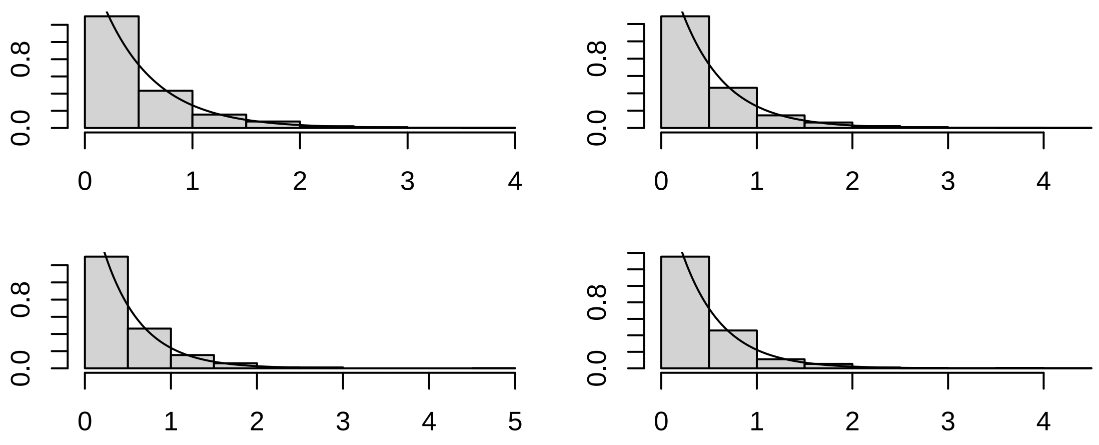


Figure 7.5: Histograms of Y values corresponding to slices of X observations, from the data simulated as in Example 7.7. The conditional PDF of X for the midpoint of slice is overlaid in each case.

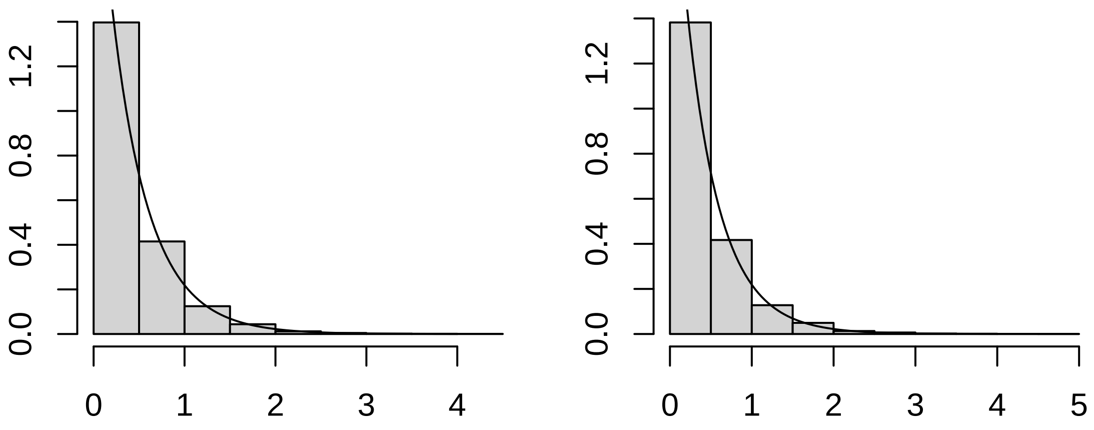


Figure 7.6: Histograms of X and Y data simulated as in Example 7.7, with overlaid marginal PDFs.

2. its m-fold integral is 1.

The most general form of joint PDF is seldom used in practice. The independence assumption is often made:

$$f_{X_1,X_2,\ldots,X_m}(x_1,x_2,\ldots,x_m) = f_{X_1}(x_1)f_{X_2}(x_2)\cdots f_{X_m}(x_m).$$

A further assumption is often made: that $f_{X_1}(x) = \ldots = f_{X_m}(x)$. In other words, the measurements $X_1, X_2, \ldots, X_m$ are assumed to follow the same distribution. When handling time series data, these assumptions are usually false, but they can be used to build models where there are realistic forms of dependence. The next sections and chapters explore these ideas.

7.10 Maximum likelihood estimation

Until this point, we have considered a variety of models, many of which are governed by parameters, such as means, shapes, scales, rates, and so on, but we have not indicated how these parameters are chosen in practice. The best practice is to estimate them from data, and for that purpose, maximum likelihood estimation is often, but not always, the preferred method of estimation.

7.10.1 The joint PDF is the likelihood function

Suppose m measurements are taken from a population which follows a joint PDF of the form

$$f_{X_1,X_2,\ldots,X_m}(x_1, x_2, \ldots, x_m)$$

but where there are one or more unknown parameters, which we might denote with the symbol θ. That means that the joint PDF is really also a function of θ:

$$f_{X_1,X_2,\ldots,X_m}(x_1, x_2, \ldots, x_m, \theta).$$

Since the m measurements are observed, their values are known, but θ is not known, so the function actually varies with θ but no longer with the measurements, since their values are fixed. We can thus view the function in the following way:

$$L(\theta; x_1, x_2, \ldots, x_m) = f_{X_1,X_2,\ldots,X_m}(x_1, x_2, \ldots, x_m, \theta).$$

The function $L(\theta)$ is referred to as the likelihood function. It is a function of θ and it turns out to be very useful in helping to choose a good value of θ, when measurements have been observed. Specifically, recall that a PDF has large values in regions where the random variables have a higher probability of taking values and lower values in regions where there is low probability of occurrence. Therefore, we would want to choose θ to ensure that the joint PDF takes large values, and in fact, the largest possible values. In other words, we would want to choose θ to maximize $L(\theta)$ with respect to θ.

Example 7.11 *Suppose X and Y have been observed as $x = 3$ and $y = 7$, and X and Y come from a population with joint PDF given by*

$$f_{X,Y}(x, y) = \lambda^2 e^{-\lambda(x+y)}, \quad \textit{for } x > 0, y > 0$$

and 0, otherwise. This means that the likelihood function $L(\lambda)$ is

$$L(\lambda) = \lambda^2 e^{-10\lambda}.$$

The function is plotted in Figure 7.7, and the maximum appears to be near 0.2.

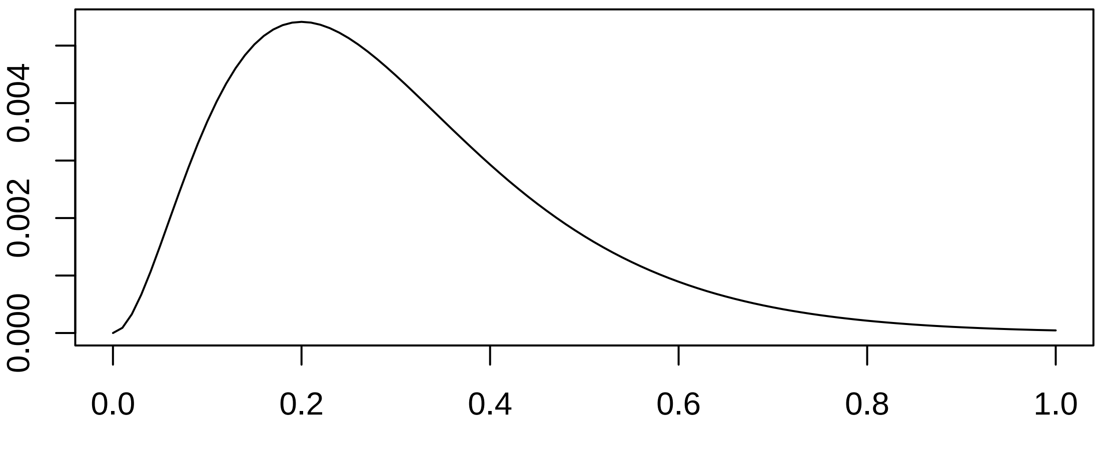


Figure 7.7: Likelihood function for Example 7.11.

Calculus can often, though not always, be employed to obtain the maximum value, the so-called maximum likelihood estimator. In Example 7.11, one would differentiate the function $L(\lambda)$ (or $\log(L(\lambda))$, since it is simpler to use), to obtain a function of λ which would be set to 0 and solved. The result in this case can be found to be $\widehat{\lambda} = 0.2$.

7.10.2 Maximum likelihood estimation with independent, identically distributed measurements

The likelihood function for independent measurements $x_1, \ldots, x_n$ coming from a population modelled by a PDF $f(y)$ is

$$L(\theta) = f(x_1)f(x_2)\cdots f(x_n)$$

where θ denotes parameter(s) to be estimated. The procedure to obtain the maximum likelihood estimator is then to maximize $\log(L(\lambda))$ with respect to θ. This is often accomplished by setting the derivative of the log likelihood equal to 0 to solve for $\widehat{\theta}$. It may be necessary to compute partial derivatives, if the number of parameters to be estimated exceeds one.

Example 7.12 *An important property of air bags is permeability of the woven fabric. This is related to their ability to absorb energy. It is important to know whether permeability is different for different temperatures.*

To determine whether temperature has an effect, we consider measurements of air bag permeability at 2 different temperatures: $0°C$ and $20°C$:

```
 0:    70, 85, 92, 80, 60
20:    40, 60, 50, 45
```

Let us model permeability at each temperature with normal distributions. Denote the expected value of permeability at $0°$ by μ_0 and at $20°$ by μ_{20}. Let the variance be σ^2 in both cases. We are assuming that variability is the same at different temperatures. This can be informally checked with a side-by-side boxplot. How do we estimate μ_0, μ_{20} and σ^2? Let $X_1, x_2, \ldots, X_m$ denote the 1st sample, and let $Y_1, Y_2, \ldots, Y_n$ denote the 2nd sample. Note that $m = 5$ and $n = 4$ for our samples.

The PDF for one of the X measurements is

$$f(x) = \frac{e^{-(x-\mu_0)^2/(2\sigma^2)}}{(2\pi\sigma^2)^{1/2}}$$

The PDF for one of the Y measurements is

$$f(x) = \frac{e^{-(x-\mu_{20})^2/(2\sigma^2)}}{(2\pi\sigma^2)^{1/2}}$$

The likelihood function is

$$L(\mu_0, \mu_{20}, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{9/2}} e^{-\sum_{i=1}^m (x_i-\mu_0)^2/(2\sigma^2) - \sum_{j=1}^n (y_j-\mu_{20})^2/(2\sigma^2)}$$

$$\log L = -\frac{\sum_{i=1}^m (x_i-\mu_0)^2}{2\sigma^2} - \frac{\sum_{j=1}^n (y_j-\mu_{20})^2}{2\sigma^2} - 9/2\log(2\pi\sigma^2)$$

Differentiating with respect to μ_0 and setting to 0 gives

$$\widehat{\mu}_0 = \frac{1}{m}\sum_{i=1}^m X_i = \bar{X}$$

Differentiating with respect to μ_{20} gives

$$\widehat{\mu}_{20} = \frac{1}{n}\sum_{j=1}^n Y_j = \bar{Y}$$

Also,

$$\widehat{\sigma}^2 = \frac{\sum_{i=1}^m (X_i - \bar{X})^2 + \sum_{j=1}^n (Y_j - \bar{Y})^2}{n+m}$$

Plugging the data into these estimation formulas gives: $\bar{x} = 77.4$, $\bar{y} = 48.8$, and $\widehat{\sigma}^2 = 122$

7.11 General results concerning collections of measurements

7.11.1 Expectation of a function of several random variables

From the definition of expectation for pairs of random variables, we can derive a simple way to find the expected value of a sum of random variables: sum the expected values as in

$$\begin{aligned}
\mathrm{E}[X_1 + X_2] &= \int\int (y_1 + y_2) f(y_1, y_2) dy_1 dy_2 \\
&= \int\int y_1 f(y_1, y_2) dy_1 dy_2 \\
&+ \int\int y_2 f(y_1, y_2) dy_1 dy_2 \\
&= \mathrm{E}[X_1] + \mathrm{E}[X_2].
\end{aligned}$$

When there are three random variables, and we wish to calculate the expected value of their sum, we can employ a triple integral and a simplification corresponding to what we see above occurs, so that

$$\mathrm{E}[X_1 + X_2 + X_3] = \mathrm{E}[X_1] + \mathrm{E}[X_2] + \mathrm{E}[X_3].$$

Similarly,

$$\mathrm{E}[X_1 + X_2 + X_3 + X_4] = \mathrm{E}[X_1] + \mathrm{E}[X_2] + \mathrm{E}[X_3] + \mathrm{E}[X_4],$$

and so on, for sequences of any length m.

Example 7.13 *We reconsider Example 7.4. Because of contaminants in the propane, and because of interactions among the propane gas molecules, and so on, the pressure can be more accurately modelled as*

$$P = 3XT + \varepsilon$$

where ε is a random variable representing all unaccounted for factors (that is, noise). We assume $E[\varepsilon] = 0$. Find $E[P]$.

We have already seen that $E[3XT] = 568.75$, so we use the result above to conclude that

$$\begin{aligned}
E[P] = E[3XT + \varepsilon] &= E[3XT] + E[\varepsilon] \\
&= 568.75 + 0 \\
&= 568.75
\end{aligned}$$

In general, the expectations of a linear combination of a sequence of random variables is equal to the linear combination of the sequence of expected values:

$$E[\sum_{j=1}^{m} \alpha_j X_j] = \sum_{j=1}^{m} \alpha_j E[X_j],$$

where $\alpha_1, \alpha_2, \ldots, \alpha_m$ are constant values.

Example 7.14 *Suppose X_1 and X_2 are normal random variables with mean 3 and variance 9.*

1. *Find the expected value of $Y = 2X_1 - 4X_2$.*

$$E[Y] = E[2X_1 - 4X_2] = 2E[X_1] - 4E[X_2] = 6 - 12 = -6.$$

2. *Find the expected value of $W = 2X_1^2 - 4X_2^2$.*

$$E[X_1^2] = E[X_2^2] = 9 + 9 = 18 \quad (\textit{Why?})$$

Therefore,

$$E[W] = 2(18) - 4(18) = -36.$$

Example 7.15 *Measurements were taken on the amount of vibration (in microns) produced by six electric motors all having the same type of bearings. Each such measurement has been modelled with the density function*

$$f(y) = \frac{1}{10} e^{-(y-5)/10}, \quad y > 5$$

Find the expected value of the average of the 6 vibration measurements.

$$\mu = E[X_1] = \cdots = E[X_6] =$$

$$\int_5^\infty y \frac{1}{10} e^{-(y-5)/10} dy = 15$$

so that

$$E[\overline{X}] = \mu = 15.$$

7.11.2 The variance of a sum of independent random variables

Suppose X_1 and X_2 are independent random variables. Then

$$\mathrm{E}[X_1 X_2] = \int \int y_1 y_2 f_1(y_1) f_2(y_2) dy_1 dy_2 = \mathrm{E}[X_1]\mathrm{E}[X_2]$$

and

$$\mathrm{E}[(X_1 + X_2)^2] = \mathrm{E}[X_1^2] + 2\mathrm{E}[X_1 X_2] + \mathrm{E}[X_2^2]$$

so

$$\mathrm{Var}(X_1 + X_2) = \mathrm{E}[(X_1 + X_2)^2] - (\mathrm{E}[X_1 + X_2])^2 = \mathrm{E}[X_1^2] + \mathrm{E}[X_2^2] - (\mathrm{E}[X_1])^2 - (\mathrm{E}[X_2])^2$$
$$= \mathrm{Var}(X_1) + \mathrm{Var}(X_2).$$

For m independent random variables $X_1, X_2, \ldots, X_m$,

$$\mathrm{Var}(X_1 + X_2 + \cdots + X_m) =$$

$$\mathrm{Var}(X_1) + \cdots \mathrm{Var}(X_m).$$

If all of the random variables have the same variance, a further simplification occurs. Suppose $X_1, X_2, \ldots, X_m$ is a sample of independent measurements. If the variance of each is σ^2, then

$$\mathrm{Var}(X_1 + X_2 + \cdots + X_m) = m\sigma^2$$

and

$$\mathrm{Var}(\overline{X}) = \sigma^2/m$$

where

$$\overline{X} = \frac{1}{m} \sum_{k=1}^{m} X_k$$

The square root of this expression is the standard error of the mean which we have discussed earlier but without the full justification for the formula provided here.

Example 7.16 *Reconsider Example 7.15. Find the variance of the average of the 6 vibration measurements.*

$$\sigma^2 = E[X_1^2] - E[X_1]^2$$

$$E[X_1^2] = \frac{1}{10} \int_5^\infty y^2 e^{-(y-5)/10} dy = 325$$

so that

$$\sigma^2 = 325 - 15^2 = 100.$$

Therefore,

$$\mathit{Var}(\overline{X}) = 100/6 = 16.7.$$

Exercises

1. Refer to Example 7.4.

   (a) Verify that $f(x,t)$ is a valid joint density function.

   (b) Show that the probability that the amount of propane is between 10.4 and 10.6 moles, and that the temperature is between 16 and 17 degrees is 0.032.

   (c) Show that the marginal densities of X and T are

$$f_X(x) = \int_{15}^{20} \frac{x + \frac{t}{5} - 13}{5} dt =$$

$$= x - 9.5, \quad (x \in [10, 11])$$

   and

$$f_T(t) = \int_{10}^{11} \frac{x + \frac{t}{5} - 13}{5} dx =$$

$$= \frac{t}{25} - \frac{1}{2}, \quad (t \in [15, 20]).$$

2. Suppose the reliability of an electric motor depends upon two critical components, having lifetimes X and Y which can be modelled with the joint probability density function

$$f(x,y) = \begin{cases} xe^{-x(1+y)}, & x \geq 0, y \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

   (a) Show that the probability that both components last at least 1 unit of time is $P(X > 1, Y > 1) = e^{-2}/2$.

   (b) Show that the marginal density of X is

$$f_1(x) = \int_0^\infty xe^{-x(1+y)} dy = e^{-x}, \quad x \geq 0.$$

   (c) Show that the marginal density function for Y is

$$f_2(y) = \int_0^\infty xe^{-x(1+y)} dx = \frac{1}{1+y^2} \quad y \geq 0$$

   by integrating by parts.

   (d) Show that X and Y are not independent.

3. Suppose X_1, X_2, X_3 are independent normal random variables with expected values μ_1, μ_2, μ_3, and variances σ^2.

   (a) Determine the expected value and variance of $\sum_{k=1}^{3} X_k$.

   (b) Determine the expected value of

$$\frac{(X_1 - \mu_1)^2 + (X_2 - \mu_2)^2 + (X_3 - \mu_3)^2}{\sigma^2}$$

4. Suppose X_1 and X_2 are independent normal random variables with expected value μ and variance σ^2. Determine the expected value and variance of $\overline{X} = (X_1 + X_2)/2$.

5. Determine the expected value and variance of $\overline{X} = \frac{1}{n} \sum_{k=1}^{n} X_k$, if $X_1, X_2, \ldots, X_n$ are independent normal random variables with expected value μ and variance σ^2.

6. The standard error of Monte Carlo integral estimates can be estimated in exactly the same way for multiple integrals as for single integrals. Calculate the standard error of the integral estimate in Example 7.10 and use this in a 95% confidence interval.

7. Refer to Example 7.11. Differentiate $\log(L(\lambda))$ with respect to λ and show that the maximizer is $\widehat{\lambda} = 0.2$.

8. In a study of the properties of metal plate-connected trusses used for roof support, it was found that axial stiffness may depend on plate length. For 4 inch plates, measurements were 315, 387, and 358. For 12 inch plates, measurements were 401 and 460. Estimate the expected axial stiffness at each plate length and estimate the common variance σ^2.

9. Refer to Example 7.12. Suppose permeability was measured at $-20°$ as well, giving measurements 80, 85, 90, and 75. Find maximum likelihood estimates for μ_{-20}, μ_0, μ_{20} and σ^2 now.

10. In an experiment to measure acceleration due to gravity g, a number of objects of varying mass (5 kg, 10 kg, 15 kg, and 20 kg) were weighed (in Newtons): (49.5, 100.1, 147, 196). A *model* for this data is

$$W_i = m_i g + \varepsilon_i$$

where $i = 1, 2, 3, 4$ and the *measurement errors* ε_i are normally distributed with mean 0 and variance σ^2. The density of ε_i is

$$\begin{aligned} f(\varepsilon) &= \frac{e^{-(\varepsilon)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}} \\ &= \frac{e^{-(W_i - g m_i)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}} \end{aligned}$$

so the likelihood for the data is

$$L(g, \sigma^2) = \frac{e^{-\sum_{i=1}^{4}(W_i - g m_i)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}^{4}}$$

Find the maximum likelihood estimates for g and σ^2.

Bibliography

Albert, J. (2018). *LearnBayes: Functions for Learning Bayesian Inference*. R package version 2.15.1.

Anderson, S. L. (1990). Random number generators on vector supercomputers and other advanced architectures. *SIAM review*, 32(2):221–251.

Blackman, D. and Vigna, S. (2021). Scrambled linear pseudorandom number generators. *ACM Transactions Mathematical Software*, 47:1–32.

Borchers, H. W. (2021). *numbers: Number-Theoretic Functions*. R package version 0.8-2.

Braun, W. and MacQueen, S. (2019). Mpv: Data sets from Montgomery, Peck and Vining. *R package version*, 1.

Cappé, O., Moulines, E., and Rydén, T. (2005). *Inference in Hidden Markov Models*. Springer, New York.

Davies, R. and Brooks, R. J. (2007). Stream correlations in multiple recursive and congruential generators. *Journal of Simulation*, 1(2):131–135.

Eddelbuettel, D. and Francois, R. (2022). *RcppGSL: 'Rcpp' Integration for 'GNU GSL' Vectors and Matrices*. R package version 0.3.12.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis (Vol. 2)*. Taylor & Francis, Boca Raton.

Gevorkyan, M. N., Kulyabov, D. S., Demidova, A. V., and Korolkova, A. V. (2020). A practical approach to testing random number generators in computer algebra systems. *Computational Mathematics and Mathematical Physics*, 60:65–73.

Geyer, C. J. and Johnson, L. T. (2020). *mcmc: Markov Chain Monte Carlo*. R package version 0.9-7.

Han, L. and Braun, W. J. (2014). Dionysus: a stochastic fire growth scenario generator. *Environmetrics*, 25(6):431–442.

Hankin, R. K. S. (2006). Special functions in r: introducing the gsl package. *R News*, 6.

Hillier, F. and Lieberman, G. (2021). *Introduction to Operations Research, 11th Edition*. McGraw-Hill, New York.

Himmelmann, S. S. D. L. (2022). *HMM: Hidden Markov Models*. R package version 1.0.1.

James, F. (1994). Ranlux: A fortran implementation of the high-quality pseudorandom number generator of lüscher. *Computer Physics Communications*, 79(1):111–114.

Lamport, L. (1994). *LaTeX - A Document Preparation System: User's Guide and Reference Manual, Second Edition*. Pearson / Prentice Hall.

Law, A. M., Kelton, W. D., and Kelton, W. D. (2007). *Simulation Modeling and Analysis*, volume 3. Mcgraw-Hill, New York.

L'Ecuyer, P. (1996). Combined multiple recursive random number generators. *Operations research*, 44(5):816–822.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

Lüscher, M. (1994). A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications*, 79(1):100–110.

Maindonald, J. and Braun, W. (2015). Package 'DAAG'. *Data Analysis and Graphics Data and Functions*.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30.

O'Neill, M. E. (2014). Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Transactions on Mathematical Software*.

Panneton, F. and L'ecuyer, P. (2005). On the xorshift random number generators. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 15(4):346–361.

Plummer, M. (2022). *rjags: Bayesian Graphical Models using MCMC*. R package version 4-13.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ross, S. M. (2014). *Introduction to Probability Models*. Academic press.

Savage, S., Schrage, L., Lewis, P., and Empey, D. (1994). The bad seeds—a parallel random number generation problem weeded out long ago, crops up again. In *Unpublished manuscript dated 14th January 1994 (presented at the 35th TIMS/ORSA Joint National Meeting in Chicago in 1993)*.

Stubner, R. (2021). *dqrng: Fast Pseudo Random Number Generators*. R package version 0.3.0.

Sturtz, S., Ligges, U., and Gelman, A. (2005). R2winbugs: A package for running winbugs from r. *Journal of Statistical Software*, 12(3):1–16.

The GSL Team (1996-2021). *Gnu Scientific Library*.

Tijms, H. (1994). *Stochastic Models: An Algorithmic Approach*. Wiley, Chichester.

Tymstra, C., Bryce, R., Wotton, B., Taylor, S., Armitage, O., et al. (2010). Development and structure of prometheus: the canadian wildland fire growth simulation model. *Natural Resources Canada, Canadian Forest Service, Northern Forestry Centre, Information Report NOR-X-417.(Edmonton, AB)*.

Warne, D. J., Sisson, S. A., and Drovandi, C. (2021). Vector operations for accelerating expensive Bayesian computations–a tutorial guide. *Bayesian Analysis*, 1(1):1–30.

Wickham, H. (2014). *Advanced r*. CRC press.

Wood, S. N. (2015). *Core Statistics*. Cambridge University Press, New York.

Xie, Y. (2018). knitr: a comprehensive tool for reproducible research in r. In *Implementing Reproducible Research*, pages 3–31. Chapman and Hall/CRC.

Index