Modelling and Simulation in R

W. John Braun
University of British Columbia

July 4, 2021

Contents

1

An Overview of R

R is based on the computer language S, developed by John Chambers and others at Bell Laboratories in 1976. Robert Gentleman and Ross Ihaka developed an implementation, and named it R. Gentleman and Ihaka made it open source in 1995, and hundreds of people around the world have contributed to its development.

Although it may be hard for students with little mathematical or computing background to believe, R and RStudio are actually quite friendly tools, but becoming acquainted with them requires a bit of effort. A few hours of playing with R code is all that is really required to achieve modest expertise. Perhaps the most important thing to remember is that there is nothing wrong with making errors when learning a programming language like R. You learn from your mistakes, and there is no harm done. Experimentation is the key to learning R, just as it has been the key to science for the past 400 years. The reader is gently encouraged to try out the code embedded into this text and to experiment with new variations to discover how the system will respond.

1.1 Downloading and installing R and RStudio

R can be downloaded for free from `http://cloud.r-project.org`CRAN. A *binary version* is usually simplest to use and can be installed in Windows and Mac fairly easily. A binary version is available for Windows Vista or above from the web page `http://cloud.r-project.org/bin/windows/base`. The "setup program"setup is usually a file with a name like `R-3.6.1-win.exe`. Clicking on this file will start an almost automatic installation of the R system. Clicking "Next" several times is often all that is necessary in order to complete the installation. An R icon will appear on your computer's desktop upon completion.

RStudio is also very popular. You can download the "Open Source Edition" of "RStudio Desktop" from `http://www.rstudio.com/`rstudio.com, and follow the instructions to install it on your computer. Although much or all of what is described in this booklet can be carried out in RStudio, there will be little further comment about that environment. Thus, you might find that some of the instructions to be carried out at the command line can also be carried out with the menu system in RStudio.

1.2 Executing commands in R

Following installation, you should see an "R" icon on the Windows desktop or in your listing of Mac applications. Clicking on it, or opening RStudio similarly should provide you with access to a window or pane, called the R console in which you can execute commands. The > sign is the R prompt which indicates where you can type in the command to be executed.

For example, you can do arithmetic of any type, including multiplication:

```
> 1111+1234
```

By hitting the "Enter" key, you are asking R to execute this calculation.

```
1111+1234

## [1] 2345
```

Often, you will type in commands such as this into a script window, as in RStudio, for later execution, through hitting "ctrl-R" or another related keystroke sequence.

Objects that are built in to R or saved in your workspace, i.e. the environment in which you are currently doing your calculations, can be displayed, simply by invoking their name. For example, there is a data set (referred to as a data frame in R) called `women` which contains information on heights and weights of American women:

```
> women
```

```
##    height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

In the remainder of the text, we will simply type the command and corresponding output without including the > prompt symbol.

1.3 Key features of R

1.3.1 Packages

You can do many things with base R, but one of the major strengths of R is the availability of add-on packages that have been created by statisticians and computer scientists from around the world. There are literally thousands of packages, e.g. `graphics`, `ggplot2`, and `MPV`. A package contains functions and data which extend the abilities of R. Every installation of R contains a number of packages by default (e.g. `base`, `stats`, and `graphics`) which are automatically loaded when you start R.

To load an additional package, for example, called `DAAG`, type

```
library(DAAG)
```

If you get a warning that the package is can't be found, then the package doesn't exist on your computer, but it can likely be installed. Try

```
install.packages("DAAG")
```

In RStudio, it may be simpler to use the `Packages` menu.

Once `DAAG` is loaded, you can access data sets and functions that were not avaiable previously. For example, the `seedrates` data frame is now available:

```
seedrates
```

```
##   rate grain
## 1   50  21.2
## 2   75  19.9
## 3  100  19.2
## 4  125  18.4
## 5  150  17.9
```

1.3.2 Calculations in R

You can control the number of digits in the output with the `options()` function. This is useful when reporting final results such as means and standard deviations, since including excessive numbers of digits can give a misleading impression of the accuracy in your results. Compare

```
583/31

## [1] 18.80645
```

with

```
options(digits=3)
583/31

## [1] 18.8
```

Observe the patterns in the following calculations.

```
options(digits = 18)
1111111*1111111

## [1] 1234567654321

11111111*11111111

## [1] 123456787654321

111111111*111111111

## [1] 12345678987654320
```

With a few seconds of thought you will realize that R has given the incorrect value in the final calculation. This is due to the way R stores information about numbers. As is the case with most programming languages, a limited number of digits are available to store numbers, and floating-point arithmetic is used to carry out computations. In the above example, we are seeing, first hand, how many digits of numeric storage are avaiable: around 17 digits.

1.3.3 Data frames

Most data sets are stored in R as data frames, such as the `women` object we encountered earlier. Data frames are like matrices, but where the columns have their own names. You can obtain information about a built-in data frame by using the `help()` function. For example, observe the outcome to typing `help(women)`.

It is generally unwise to inspect data frames by printing their entire contents to your computer screen, as it is far better to use graphical procedures to display large amounts of data or to exploit numerical summaries. The `summary()` function provides information about the main features of a data frame:

```
summary(women)
```

```
##      height         weight
##  Min.   :58.0   Min.   :115
##  1st Qu.:61.5   1st Qu.:124
##  Median :65.0   Median :135
##  Mean   :65.0   Mean   :137
##  3rd Qu.:68.5   3rd Qu.:148
##  Max.   :72.0   Max.   :164
```

Columns can be of different types from each other. An example is the built-in `chickwts` data frame:

```
summary(chickwts)
```

```
##      weight           feed
##  Min.   :108   casein   :12
##  1st Qu.:204   horsebean:10
##  Median :258   linseed  :12
##  Mean   :261   meatmeal :11
##  3rd Qu.:324   soybean  :14
##  Max.   :423   sunflower:12
```

If you want to see the first few rows of a data frame, you can use the `head()` function:

```
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

The `tail()` function displays the last few rows. The number of rows can be determined using the `nrow()` function:

```
nrow(chickwts)
```

```
## [1] 71
```

Similarly, the `ncol()` function counts the number of columns. The `str()` function is another way to extract information about a data frame:

```
str(chickwts)
```

```
## 'data.frame': 71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

1.3.4 Reading data into a data frame from an external file

You will usually have a data set that you wish to read into R. If you have prepared it yourself, you could simply type it into a text file, for example called `mydata.txt`, perhaps with a header indicating column names, and

where you use blank spaces to separate the data entries. The `read.table()` function will read in the data for you as follows:

```
mydata <- read.table("mydata.txt", header = TRUE)
```

The object `mydata` now contains the data read in from the external file. You could use any name that you wish in place of `mydata`, as long as the first element of its name is an alphabetic character.

If the data entries are separated by commas and there is no header row, as in the file *wx_l3_2006.txt*, you would type:

```
wx1 <- read.table("wx_l3_2006.txt", header=F, sep=",")
```

Often, your data will be in a spreadsheet. If possible, export it as a `.csv` file and use something like the following to read it in.

```
wx2 <- read.table("wx_l3_fwi_2006-2011.csv", header=F, sep=",")
```

If you cannot export to `.csv`, you can leave it as `.xlsx` and use the `read.xslx()` command in the xlsx package (Dragulescu and Arendt, 2018).

Most likely, the data file that you have is not very clean in that there could be missing values or blank spaces in awkward locations, and so on. When reading in a file with columns separated by blanks with blank missing values, you can use code such as

```
dataset1 <- read.table("file1.txt", header=TRUE, sep=" ", na.string=" ")
```

This tells R that the blank spaces should be read in as missing values. Observe the contents of `dataset1`:

```
dataset1
```

```
##    x  y   z
## 1  3  4  NA
## 2 51 48  23
## 3 23 33 111
```

Note the appearance of `NA`. This represents a missing value. We note, in passing, that functions such as `is.na()` are important for detecting missing values in vectors and data frames. For more information about handling of missing values, check out the `See Also` section of `help(is.na)` and the mice package (van Buuren and Groothuis-Oudshoorn, 2011).

Sometimes, external software exports data files that are tab-separated. When reading in a file with columns separated by tabs with blank missing values, you could use code like

```
dataset2 <- read.table("file2.txt", header=TRUE, sep="\t", na.string=" ")
```

Again, observe the result:

```
dataset2
```

```
##    x   y  z
## 1 33 223 NA
## 2 32  88  2
## 3  3  NA NA
```

If you need to skip the first 3 lines of a file to be read in, use the `skip=3` argument.

1.3.5 Extracting information from data frames

To extract the `height` column from the `women` data frame, use the `$` operator:

```
women$height
```

```
##  [1] 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
```

If you want only the chicks who were fed horsebean, you can apply the `subset()` function to the `chickwts` data frame:

```
chickHorsebean <- subset(chickwts, feed == "horsebean")
chickHorsebean
```

```
##    weight      feed
## 1     179 horsebean
## 2     160 horsebean
## 3     136 horsebean
## 4     227 horsebean
## 5     217 horsebean
## 6     168 horsebean
## 7     108 horsebean
## 8     124 horsebean
## 9     143 horsebean
## 10    140 horsebean
```

You can now calculate the mean and standard deviation, and so on, of these weights:

```
mean(chickHorsebean$weight) # mean
```

```
## [1] 160.2
```

```
sd(chickHorsebean$weight)     # standard deviation
```

```
## [1] 38.626
```

In order to extract the 4th row from the `chickHorsebean` data frame, type

```
chickHorsebean[4, ]
```

```
##   weight      feed
## 4    227 horsebean
```

To extract the element in the 2nd column of the 7th row of `women`, type

```
women[7, 2]
```

```
## [1] 132
```

If we want the elements in the 4th through 7th row of the 2nd column of `women`, we can use

```
women[4:7, 2]
```

```
## [1] 123 126 129 132
```

Note the use of the `:` operator:

```
4:7

## [1] 4 5 6 7
```

Another built-in data frame is `airquality`. If we want to compute the mean for each of the first 4 columns of this data frame, we can use the `sapply()` function:

```
sapply(airquality[, 1:4], mean)

##   Ozone Solar.R    Wind    Temp
##      NA      NA  9.9575 77.8824
```

The `sapply()` function applies the same function to all columns of the supplied data frame. Note also the very useful functions in Wickham's (2011) plyr package.

1.3.6 Factors

Factors offer an alternative, often more efficient, way of storing character data. For example, a factor with 6 elements and having the two levels, `control` and `treatment` can be created using:

```
grp <- c("control", "treatment",  "control", "treatment", "treatment", "control")
grp

## [1] "control"   "treatment" "control"   "treatment" "treatment"
## [6] "control"
```

```
grp <- factor(grp)
grp

## [1] control   treatment control   treatment treatment control
## Levels: control treatment
```

Consider the built-in data frame `InsectSprays`

```
summary(InsectSprays)

##      count       spray
##  Min.   : 0.0   A:12
##  1st Qu.: 3.0   B:12
##  Median : 7.0   C:12
##  Mean   : 9.5   D:12
##  3rd Qu.:14.2   E:12
##  Max.   :26.0   F:12
```

The second column of this data frame is a factor representing the different types of spray used in the associated experiment. The levels of this factor can be listed using the `levels()` function:

```
levels(InsectSprays$spray)

## [1] "A" "B" "C" "D" "E" "F"
```

Factors are a more efficient way of storing character data when there are repeats among the vector elements. This is because the levels of a factor are internally coded as integers.

To see what the codes are for our factor, we can type

```
as.integer(InsectSprays$spray)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
## [34] 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6
## [67] 6 6 6 6 6 6
```

The labels for the levels are only stored once each, rather than being repeated. We can change the labels for the factor using the `levels()` function as follows:

```
levels(InsectSprays$spray)[3] <- "Raid"
```

Observe the effect of the change in

```
summary(InsectSprays$spray)
```

```
##    A    B Raid    D    E    F
##   12   12   12   12   12   12
```

The `levels()` function also offers a simple way to collapse categories. Suppose we are interested in comparing the first three levels with the last three levels. We can create a new factor for this purpose as follows:

```
InsectSprays$newFactor <- InsectSprays$spray
levels(InsectSprays$newFactor) <- c("A", "A", "A", "B", "B", "B")
```

Check the result:

```
summary(InsectSprays)
```

```
##       count       spray    newFactor
##  Min.   : 0.0   A   :12   A:36
##  1st Qu.: 3.0   B   :12   B:36
##  Median : 7.0   Raid:12
##  Mean   : 9.5   D   :12
##  3rd Qu.:14.2   E   :12
##  Max.   :26.0   F   :12
```

1.3.7 Histograms and simulated normal data

The `hist()` function can be used to draw histograms, and the `rnorm()` function can be used to simulate draws from a normal distribution[1]

A standard normal random variable has a mean of 0 and a standard deviation of 1. Figure 1.1 shows the results from simulating 2000 standard normal variates, together with a plot of the normal probability density curve, obtained from the `dnorm()` function. Note that we have used the `curve()` function with the `add` argument to overlay the curve. The `col` parameter controls the colour.

```
Z <- rnorm(2000)
hist(Z, prob = TRUE)
curve(dnorm(x), from = -3, to = 3, add = TRUE, col = "blue")
```

[1] often used as a model for noise

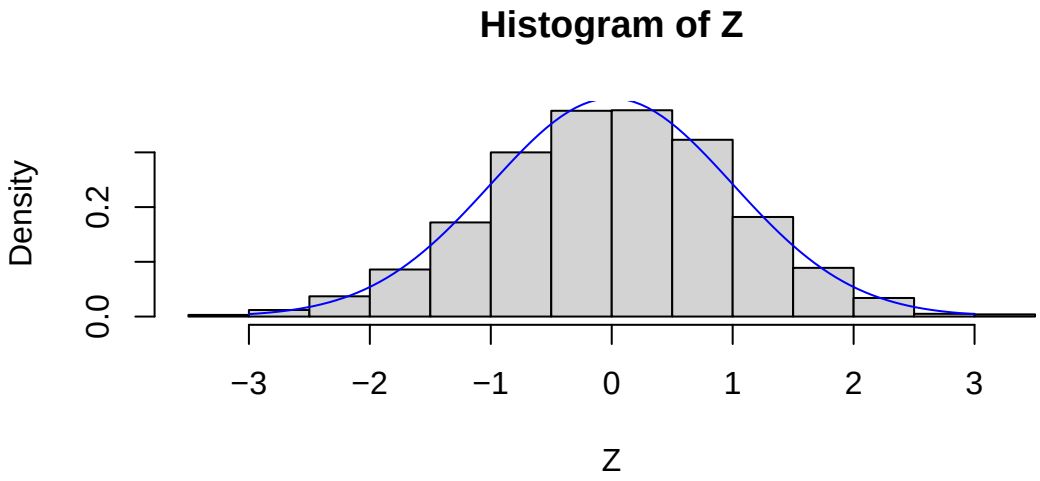


Figure 1.1: Histogram of 2000 standard normal random variates with overlaid density curve (in blue).

1.4 Flow control in R

There are several functions that control how many times statements are repeated. We will describe the `for()` and `if()` functions here.

1.4.1 The `for()` *function*

The `for()` function allows us to repeat a command a specified number of times.

Syntax:

```
for (i in indices) {commands}
```

This sequentially sets a variable called `i` equal to each of the elements of `indices`. For each value of `i`, the listed commands are executed.

Example 1.1 *We can add the elements of a vector using the* `sum()` *function, but if we want to add up a sequence of vectors, we might do it with a for loop.*

Suppose we want to simultaneously add $1 + 2 + 3 + \ldots + 100$ *and* $1^2 + 2^2 + \cdots + 100^2$. *In other words, we want to add vectors of the form* $[i \quad i^2]$, *for* $i = 1, 2, \ldots, 100$. *We will store our result in a vector called* `sums`, *and we will start by assigning* $[0 \quad 0]$ *to* `sums` *and sequentially adding vectors* $[1 \quad 1]$, $[4 \quad 4]$, *and so on:*

```
sums <- c(0, 0)
for (i in 1:100) {
     sums <- sums + c(i, i^2)
}
sums

## [1]    5050 338350
```

Example 1.2 *Simulating normal random variables is possible in a variety of ways. If we add up 12 uniform random variables on* $[-.5, .5]$, *we can get a sum that follows a close approximation to the standard normal distribution. We will use a* `for()` *loop to construct a large vector of such values so that we can draw a histogram*

and QQ-plot, to verify that we have succeeded in simulating normal random variables. We initially assign 0 to our outcome vector `Z`*. Then we successively add a uniform vector of size* $N = 10000$ *to* `Z`*, 12 times.*

```
Z <- 0; N <- 10000
for (i in 1:12) {
    U <- runif(N, min=-.5, max=.5)
    Z <- Z + U
}
```

The histogram and QQ-plot of these simulated data are given in Figure 1.2, the result of executing the following code.

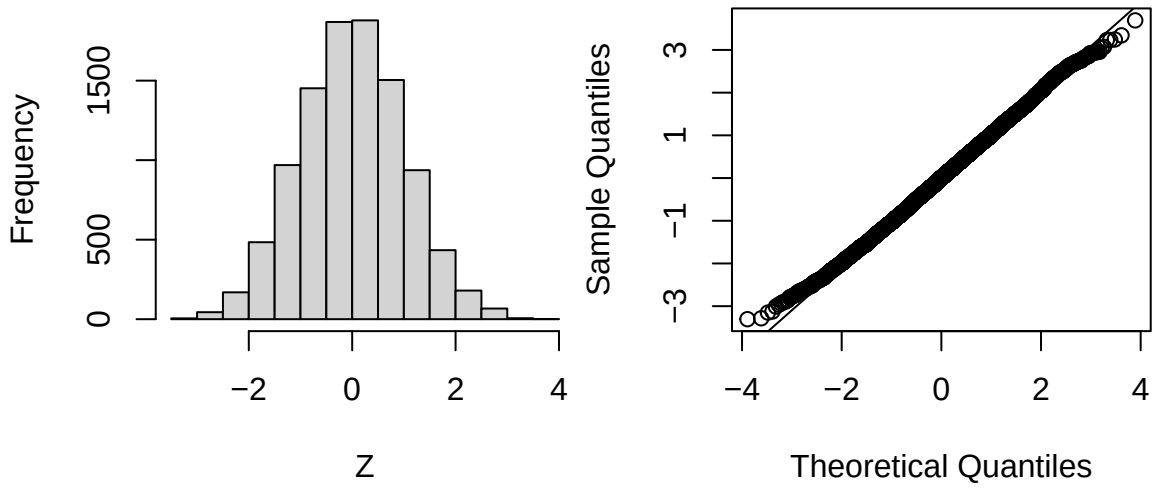


Figure 1.2: Histogram and QQ-plot of the data simulated from sums of independent uniform random variates.

```
par(mfrow=c(1,2), mar=c(4, 4, .1, .1))
hist(Z)
qqnorm(Z); qqline(Z)
```

In theory, the mean of random variables like Z *should be 0, and the standard deviation should be 1. In fact, for our simulated sample, the values of the sample mean and standard deviation are:*

```
mean(Z)   # sample average

## [1] 0.00043345

sd(Z)     # sample standard deviation

## [1] 1.0101
```

Different samples would have slightly different means and standard deviations, but all would be pretty close to 0 and 1.

Example 1.3 *Summing squared standard normal variables gives chi-squared random variables. If* Z *is a standard normal random, then* $X = Z^2$ *is called a chi-squared random variable on 1 degree of freedom.*

The distribution of a sample of chi-squared random variates on 1 degree of freedom is pictured in Figure 1.3, the effect of executing the following code:

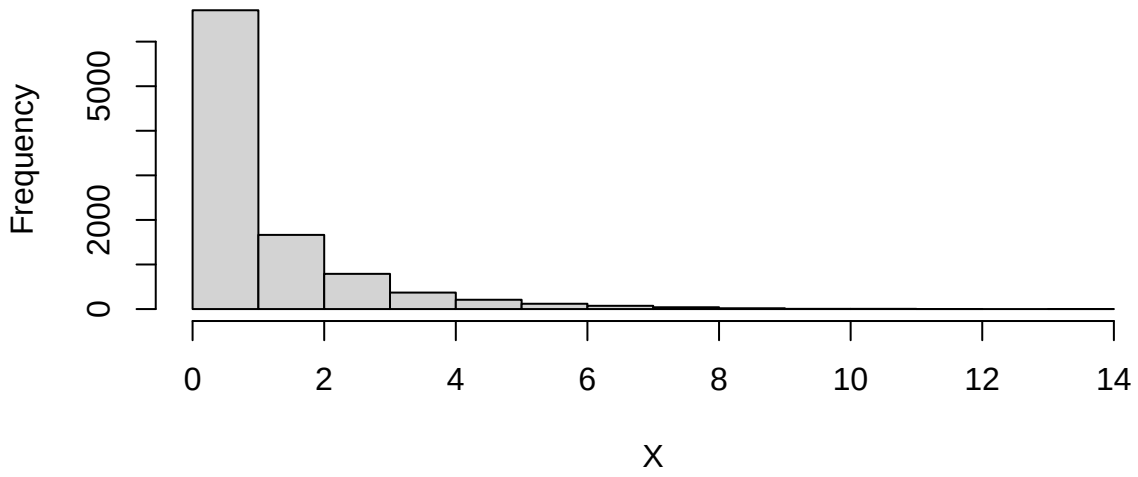


Figure 1.3: Histogram of chi-square variate on 1 degree of freedom.

```
X <- Z^2;  hist(X)
```

T is an example of a skewed distribution. Most of the values are near 0, but there are a few very large values. If $Z_1, Z_2, \ldots, Z_k$ are independent standard normal random variables, then the sum of their squares is a chi-squared random variable on k degrees of freedom.

Example 1.4 *We can use nested* `for()` *loops to simulate these sums of squared normals. For example, suppose $k = 7$ as in the following:*

```
X <- 0
for (i in 1:7) {
    Z <- 0
    for (j in 1:12) {
        U <- runif(N, min = -.5, max = .5)
        Z <- Z + U # Z is standard normal
    }
    X <- X + Z^2  # X is chi-squared
}
```

Figure 1.4 shows what a chi-squared distribution on 7 degrees of freedom looks like. It is skewed, but not as much as when the number of degrees of freedom is smaller.

```
par(mar=c(4, 4, .1, .1))
hist(X, main="")
```

1.5 The `if()` statement

The `if()` statement allows us to control which statements are executed.

Syntax:

```
if  (condition) {commands when TRUE}
if  (condition) {commands when TRUE} else {commands if FALSE}
```

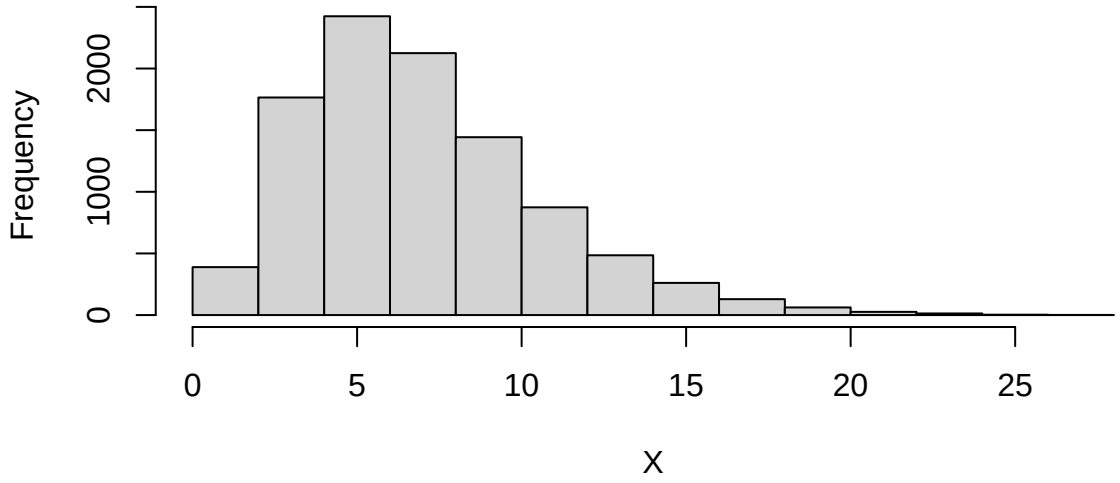


Figure 1.4: Histogram of simulated chi-square variates on 7 degrees of freedom.

This statement causes a set of commands to be invoked if `condition` evaluates to `TRUE`. The `else` part is optional, and provides an alternative set of commands which are to be invoked in case the logical variable is `FALSE`.

1.5.1 The `if()` *statement: Caution!*

Be careful how you type the `else` statement. Typing it as

```
if  (condition) {commands when TRUE}
else {commands when FALSE}
```

may produce an error, because R will execute the first line before you have time to enter the second. If these two lines appear within a block of commands in curly brackets, they won't trigger an error, because R will collect all the lines before it starts to act on any of them. To avoid this kind of difficulty, use the form

```
if  (condition) {
    commands when TRUE
} else {
    commands when FALSE
}
```

1.5.2 The `if()` *statement: Another Warning*

R also allows numerical values to be used as the value of `condition`. These are converted to logical values using the rule that zero becomes `FALSE`, and any other value becomes `TRUE`. Missing values are not allowed for the condition, and will trigger an error.

Example 1.5 `x <- 3`
`if (x > 2) y <- 2 * x else y <- 3 * x`

Since `x > 2` *is* `TRUE`, `y` *is assigned* `2 * 3 = 6`. *If it hadn't been true,* `y` *would have been assigned the value of* `3 * x`.

1.6 Functions

As we have seen, R calculations are carried out by functions, and graphs are produced by functions.

The usual composition of a function is

- a header that includes the word `function` and an argument list (which might be empty)
- a body which includes a set of statements enclosed in curly brackets `{}`.

Function names should be chosen to describe the action of the function. For example, `median()` computes medians, and `boxplot()` produces box plots.

Example 1.6 *We will write a function to approximately simulate standard normal random variables. An appropriate header for the function could be:*

```
rStdNorm <- function(n)
```

Note that this function will take `n` *as an input. The output should be that number of standard normal variates.*

At some point in the body of the function there is normally a statement like `return(Z)` which specifies the output value of the function. If there is no `return()` statement, then the value of the last statement executed is returned.

Example 1.7 *In our standard normal simulator, we will want to return a vector of length* `n`*. We will use* `Z` *as the name of this object.*

```
rStdNorm <- function(n) {
...
return(Z)
}
```

Using the sum of uniforms concept from an earlier example, we will use a function body of the form:

```
{
    Z <- 0
    for (j in 1:12) {
        U <- runif(n, min = -.5, max = .5)
        Z <- Z + U
    }
    return(Z)
}
```

Putting the header and body together, we have the following function:

```
rStdNorm <- function(n) {
    Z <- 0
    for (j in 1:12) {
        U <- runif(n, min = -.5, max = .5)
        Z <- Z + U
    }
    return(Z)
}
```

A trial with 3 values is executed as follows:

```
rStdNorm(3)

## [1]  0.77117 -1.20665 -2.25186
```

Functions may take any number of arguments.

Example 1.8 *We can use our new* `rStdNorm()` *function inside a function which calculates chi-squared random variables on* k *degrees of freedom. Two arguments,* `n` *and* `k` *will be needed in this function.*

```
rChisq <- function(n, k) {
    X <- 0
    for (i in 1:k) {
        Z <- rStdNorm(n)
        X <- X + Z^2
    }
    return(X)
}
```

A trial with $k = 17$ *degrees of freedom, and 2 values is executed as follows:*

```
rChisq(2, 17)

## [1] 22.044 15.749
```

To give the user of a function a hint as to the kind of input that the function is expecting, we may give default values to some arguments: if the user doesn't specify the value, the default will be used.

Example 1.9 *We could have used the header, i.e. the first line of the function,*

```
rChisq <- function(n, k = 1)
```

to indicate that if a user called `rChisq(10)` *without specifying* `k`*, then it should act as though* `k = 1`.

We conclude our brief discussion of functions with a mention of the function's environment. We won't give a complete description here, but will limit ourselves to the following circular definition: the environment is a reference to the environment in which the function was defined. This has implications for where objects are that the function can access. Consider the following example.

Example 1.10 *A function* `myfun` *is created in an environment that does not contain* `mydata`*:*

```
myfun <- function() {
      mymean <- mean(mydata)
      return(mymean)
}
myfun() # execute function

## Error in mean(mydata): object 'mydata' not found
```

Now, consider what happens when `mydata` *is in the function's environment:*

```
mydata <- rChisq(4, 1)
myfun()  # mydata exists now and mymean exists internally to myfun

## [1] 0.19957
```

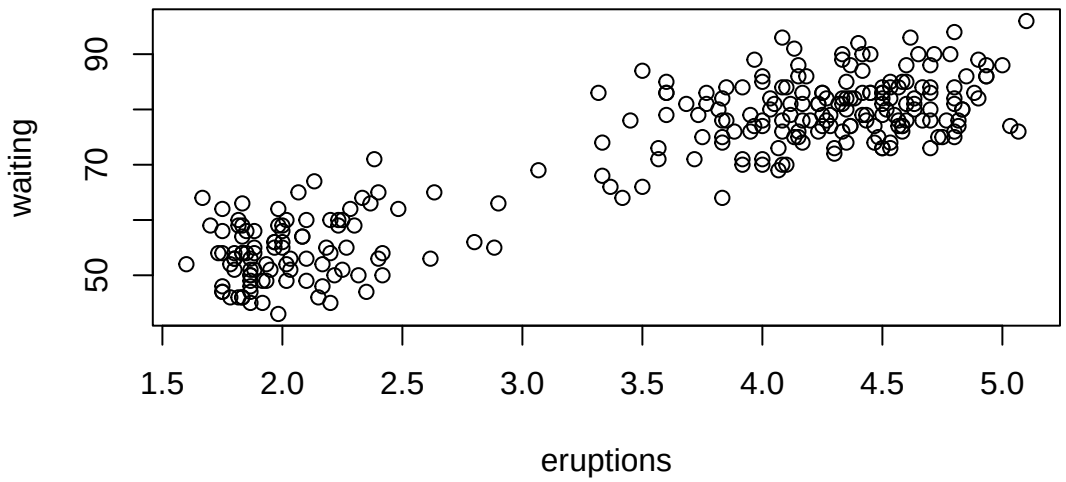
```
mymean # does not exist in the workspace, only locally to myfun
```

```
## Error in eval(expr, envir, enclos): object 'mymean' not found
```

Exercises

1. Consider the `faithful` data set that is built in to R. It consists of the waiting times until the next eruption of the Old Faithful geyser in Yellowstone National Park together with the corresponding eruption times. We might want to predict the waiting time until the next eruption by noting what the current eruption time is. The scatterplot for these data can be obtained by typing

```
plot(faithful)
```



A simple way to make predictions from such data is to smooth the scatterplot of the y values that are plotted against the x values. One way to do this is to use moving averages. In other words, just take averages of y values that are near each other according to their x values. Join these averages together to form a curve.

In this exercise, you will write a function which outputs a new data frame consisting of a column of equally spaced x values and a column of corresponding local averages, and which takes the following arguments

- `x`: the vector of x values
- `y`: the vector of y values
- `x.min`: a constant which specifies the left boundary of the plotted curve
- `x.max`: a constant which specifies the right boundary of the plotted curve
- `window`: a constant which specifies the range of the x values used to calculate each of the moving averages

(a) Write down the header for this function, assuming that the function will be called `smoother`.

```
smoother <- function(x, y, x.min, x.max, window) {
```

(b) The output for this function will be a data frame with 2 columns: `x` and `y`, which will correspond to the y-averages and the corresponding x locations where the averages are taken. Thus, include a line such as the one at the end of the following body-less function:

```
smoother <- function(x, y, x.min, x.max, window) {
    ...
    data.frame(x = xpoints, y = yaverages)
}
```

(c) Now, you need to construct the body of the function.

i. Use the `seq()` function to create a sequence of 401 equally spaced x values, starting at `x.min` and ending at `x.max`. In your function, include a line of code that assigns this sequence to an object called `xpoints`.

```
smoother <- function(x, y, x.min, x.max, window) {
    xpoints <- seq(x.min, x.max, len=401)
    ...
    data.frame(x = xpoints, y = yaverages)
}
```

ii. Use a `for()` loop to calculate the column of corresponding `yaverages`. To do this, you need to first initialize the `yaverages` object to have the same number of elements as `xpoints`. Include the following line in your function:

```
    yaverages <- numeric(length(xpoints))
```

next, for each value of i, running from 1 through `xpoints`, you need to determine which elements of the original data vector `x` are close to `xpoints[i]`, so that you can take the average of the corresponding `y` values only. In other words, you want to determine the `indices` of `x` for which the absolute value of `x - xpoints[i]` is less than the `window` parameter that was specified in the argument to the `smoother()` function you are writing.

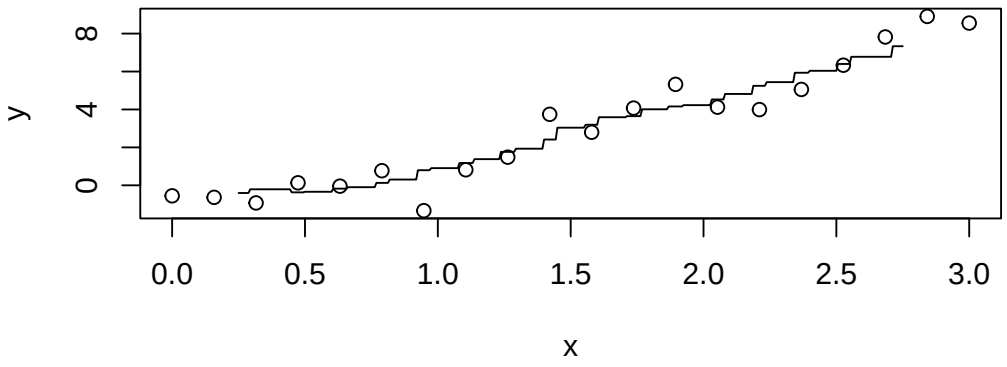
```
smoother <- function(x, y, x.min, x.max, window) {
    xpoints <- seq(x.min, x.max, len=401)
    yaverages <- numeric(length(xpoints))
    for (i in 1:length(xpoints)) {
        indices <- which(abs(x - xpoints[i]) < window)
    }
    data.frame(x = xpoints, y = yaverages)
}
```

iii. Within the `for()` loop that you just created, add a line of code which assigns the average of the values in `y[indices]` to `yaverages[i]`.

```
smoother <- function(x, y, x.min, x.max, window) {
    xpoints <- seq(x.min, x.max, len=401)
    yaverages <- numeric(length(xpoints))
    for (i in 1:length(xpoints)) {
        indices <- which(abs(x - xpoints[i]) < window)
        yaverages[i] <- mean(y[indices])
    }
    data.frame(x = xpoints, y = yaverages)
}
```

(d) You should now have a working function, provided you have not made any errors. Test out your function on some artificial data. For example, you might try something like

```
x <- seq(0, 3, length=20)
y <- x^2 + rnorm(20) # a noisy parabola
plot(x, y) # produce the scatterplot
lines(smoother(x, y, x.min=0.25, x.max=2.75, window=0.5))
```



Try different values of the `window` parameter, and in particular, see what happens when `window` is very close to 0. You should see missing pieces in your smooth curve. Why?

If the `window` parameter is too close to 0, there will be no data points close enough to some of the values in `xpoints`, so you will be averaging no data, thus, there is nothing to plot.

(e) To avoid such a problem, you can include an error message in your function to tell the user that the `window` parameter is too small. The `stop()` function provides such a message and aborts execution of the function. Within the `for` loop to your function, include the following lines of code to incorporate this:

```
  if (length(indices) < 1) {
              stop("Your choice of window width is too small.")
        } else {
              yaverages[i] <- mean(y[indices])
        }
```

```
smoother <- function(x, y, x.min, x.max, window=1) {
     xpoints <- seq(x.min, x.max, len=401)
     yaverages <- numeric(401)
     for (i in 1:length(xpoints)) {
          indices <- which(abs(x - xpoints[i]) < window)
          if (length(indices) < 1) {
              stop("Your choice of window width is too small.")
          } else {
              yaverages[i] <- mean(y[indices])
          }
     }
```

```
    data.frame(x = xpoints, y = yaverages)
}
```

(f) Finally, you should have observed that the so-called "smooth" curve is still quite bumpy. To reduce the bumpiness, we can iterate the smoothing procedure. In other words, we can repeat the smoothing procedure on the output from `smoother()`, as follows:

```
output1 <- smoother(x, y, 0.25, 2.75,  window = .5)
output2 <- smoother(output1$x, output1$y, 0.25, 2.75,  window = .25)
```

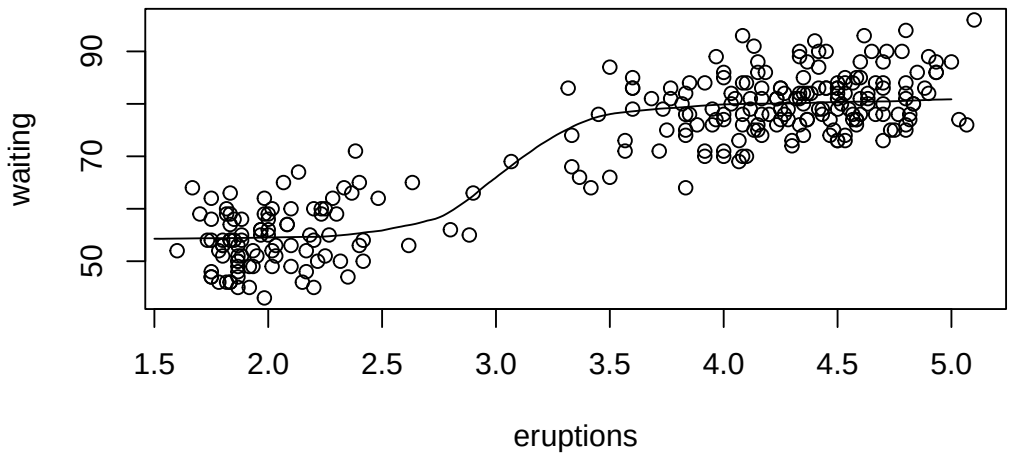
Observe that the `window` parameter does not have to be the same for each iteration.

Write a new function called `doublesmoother()` which takes the same arguments as `smoother`, but where `window` is now assumed to be a vector with 2 elements. The output from `doublesmoother()` should be a data frame consisting of `xpoints` and `yaverages` as in `smoother()` but should be the result of the second round of smoothing.

```
doublesmoother <- function(x, y, x.min, x.max, window) {
    output1 <- smoother(x, y, x.min, x.max,  window[1])
    output2 <- smoother(output1$x, output1$y, x.min, x.max,  window[2])
    output2
}
```

(g) Apply the `doublesmoother()` function to the `faithful` data frame. Use a `window` parameter of 1 unit for the first level of smoothing and a value of 0.1 unit for the second level. Use equally spaced `xpoints` in the interval $[1.5, 5.0]$. Note that the x values in this example are obtained using `faithful$eruptions`. Overlay a scatterplot of the original data with your smooth curve.

```
plot(faithful)
lines(doublesmoother(faithful$eruptions, faithful$waiting,
    1.5, 5.0, c(1, 0.1)))
```



(h) Based on your plot, make a prediction about the waiting time for the next eruption if the previous eruption took 3.25 minutes.

According to the smooth curve, which is an estimate of the expected waiting time to the next eruption for different eruption times, if the last eruption took 3.25 minutes, we would expect the next eruption to take place between 70 and 80 minutes later. We can also use the output from `doublesmoother()` to give a point estimate as follows:

```
faithful.out <- doublesmoother(faithful$eruptions,
     faithful$waiting, 1.5, 5.0, c(1, 0.1))
indices <- which(abs(faithful.out$x - 3.25) < .01) # find
  #rows of the output data frame which are close to x = 3.25
faithful.out[indices, ] # display these rows

##           x      y
## 200 3.2412 73.425
## 201 3.2500 73.662
## 202 3.2588 73.887
```

From this output, our estimate of the expected waiting time until the next eruption is 73.66 minutes.

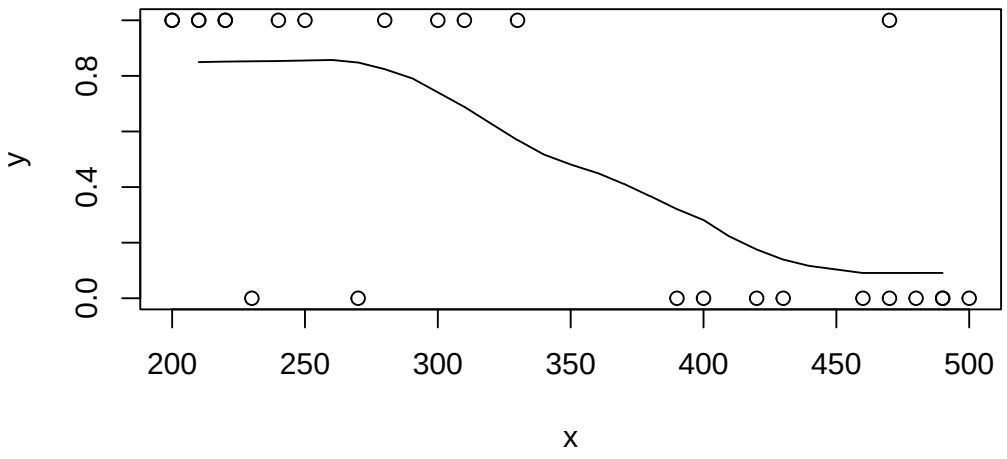
2. The `p13.1` data frame in the *MPV* library is concerned with the successes (hits) or failures (misses) of surface-to-air missiles which are supposed to hit moving targets, and it consists of the following two columns:

```
x:    target speed (in Knots)
y:    hit (=1) or miss (=0)
```

(a) Load the *MPV* library and apply the `doublesmoother()` function to the `p13.1` data frame. Use a `window` parameter of 100 units for the first level of smoothing and a value of 30 units for the second level. Use equally spaced `xpoints` in the interval $[210, 490]$. Note that the x values in this example are obtained using `p13.1$x`. Overlay a scatterplot of the original data with your smooth curve.

```
library(MPV)
```

```
plot(y ~ x, data = p13.1)
lines(doublesmoother(p13.1$x, p13.1$y, 210, 490, window=c(100, 30)))
```



(b) Based on your output, make a rough guess as to the expected proportion of times the missile would hit a target moving at a speed of 350 knots.

According to the graphed curve, the value of y is near 0.5, when $x = 350$, so we would estimate the expected proportion of hits to be 0.5.

1.7 Sources of additional information

John Maindonald has written a comprehensive introduction and overview of R which is a very useful reference for scientists. It can be found at
`https://www.researchgate.net/publication/228702931_The_R_System-An_Introduction_and_Overview`

A handy reference card has been constructed by Jonathan Barron and is available at
`http://www.psych.upenn.edu/~baron/refcard.pdf`

2

An Overview of Statistical Modelling

To a lot of people, the field of statistics can sound frightening, since the underlying theory is based on some sophisticated mathematics which is not easily understood. When simplified to the point where most people can quickly and easily understand, it is then viewed as somewhat boring or unimaginative.

In reality, statistics is an important and powerful discipline which combines elements of art and science. It lies at the heart of the scientific method, since it refines the beliefs of an investigator who brings a certain level of knowledge (including possible errors in judgement) about a scientific problem. It does this by allowing the investigator to incorporate new information in the form of data, which may or may not be in numeric form. By appropriate use of probability, the level of uncertainty in the conclusions is measured, either through confidence intervals or p-values, or using a fully probabilistic approach referred to as Bayesian. The latter approach will not be pursued here, although not because it is not important in its own right.

2.1 Types of data

Before taking measurements or observations on some type of phenomenon, they are unknown. A useful way of coping with this lack of knowledge is based on probability. Probability can allow us to quantify our uncertainty about measurements. For example, before throwing a six-sided die, we know that the number of spots that we will observe follows a specific probability distribution, and we refer to that number as a random variable, which we might refer to as Y, and we can say that the probability that $Y = 4$ is $1/6$, and that the probability that $Y = 7$ or $Y = 1.5$ is 0.

The number of spots on the die, Y, is an example of a count, a type of numeric variable. The number of heads, H, in one toss of a coin is another example of a count, but this time with only two possibilities $H = 0$ or $H = 1$. H is an example of a binary random variable or indicator variable. If you think about it, the numbers 0 or 1 are not actually observed, but rather the head or tail. Therefore, the data is, strictly speaking, not of the form of a numeric variable in this case, but rather a categorical variable, with levels Head and Tail. By using the random variable H, we have converted the categorical variable to numeric by a particular type of coding, but note that the coding was arbitrary, since we could have also defined T to be 0 or 1, depending on the number of tails observed.

Other forms of categorical data are possible as well, such as eye-colour, which might include black, brown, blue, and other. In this case, we would do the numeric coding using three binary variables, B_1, B_2 and B_3, where B_1 is 1 if the eye-colour is black, and 0, otherwise. $B_2 = 1$ for a brown eye and $B_2 = 0$, otherwise. $B_3 = 1$ for a blue eye and $B_3 = 0$, otherwise. All other eye colours are coded automatically as $B_1 = B_2 = B_3 = 0$.

Another important type of data is continuous data. Continuous variables take on measurements that are not necessarily counting numbers, and are expressed as decimals. Temperature, height, weight and time are often thought of as examples of continuous variables. An important distribution for continuous variables is the normal distribution which gives the familiar symmetric bell-shaped curve. Theoretically, normal random variables can take on any kind of value, positive and negative, so there are situations where this is clearly not appropriate. Time-to-event data, such as the time until someone recovers from a disease, or the time until a lightbulb fails, is a data type which is continuous and where the normal distribution is usually not a good approximation. Sometimes a transformation, such as a log-transformation or a square root transformation yields a new version of the variable which is better approximated by normality.

2.2 Graphic and numeric summaries

An important facet of statistics is univariate analysis, whereby the distribution of a given single random variable is studied, often through the use of summary statistics, such as the mean, median, standard deviation and so on, or through the use of graphics, such as the box plot, histogram or dot chart. A bar chart is the most effective way of conveying categorical data; although pie charts are popular, they have been largely discredited as effective data analysis tools, and should be avoided.

Consider the following examples.

2.2.1 Lake Huron levels

The `LakeHuron` object contains annual measurements of the sea level height of Lake Huron (at a particular location). A quick numeric summary of these data is obtained through

```
summary(LakeHuron)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   576.0   578.1   579.1   579.0   579.9   581.9
```

A box plot, as shown in the left panel of Figure 2.1, can be constructed using

```
par(mfrow=c(1,2), mar = c(1, 3, 1, 1))
boxplot(LakeHuron)
hist(LakeHuron)
```

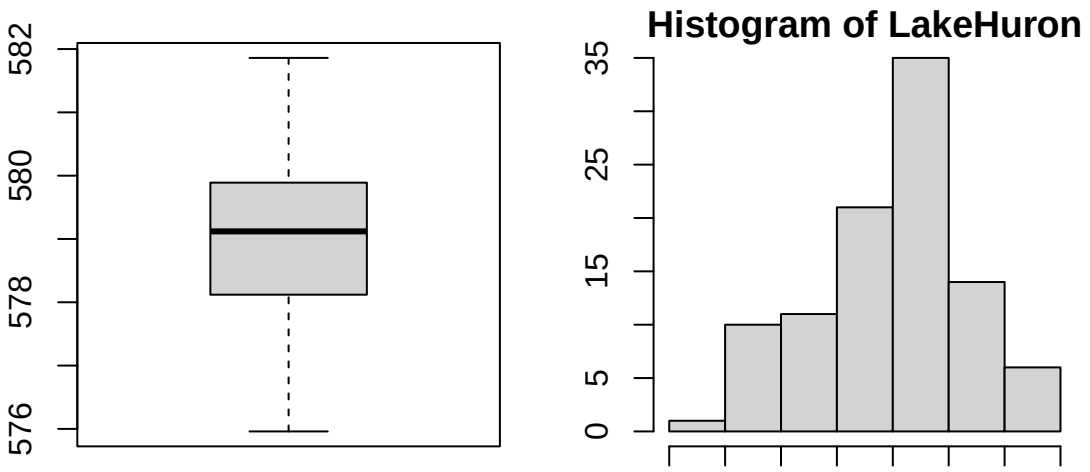


Figure 2.1: Box plot and histogram of Lake Huron lake levels.

```
summary(LakeHuron)
```

A histogram can be constructed using `hist` in place of `boxplot` and is shown in the right panel of Figure 2.1. Both plots reveal information about the marginal distribution of the lake levels. The result is not quite symmetric but indicates a degree of skewness to the left; the histogram would allow us to clearly distinguish these data from a normal distribution, if we were to assume that the measurements are independent of each other.

```
hist(LakeHuron)
```

In fact, there are some complexities underlying these data that really should be taken into account before making inferences or conclusions based on simple graphs such as we have produced here. The measurements have been collected in time sequence, and as such, should be analyzed using time series models. The assumption of independence, mentioned in the preceding paragraph, turns out to be invalid here.

2.2.2 Eye Colour - Categorical Data

A sample of brown-haired males revealed the following eye colour counts:

black	brown	blue	green
53	50	25	15

The table above provides the best form of numeric summary for this kind of data. Converting to percentages is an equivalent alternative – which hides the total number of data points.

The bar chart is constructed using

```
barplot(c("black" = 53, "brown" = 50, "blue" = 25, "green" = 15))
```

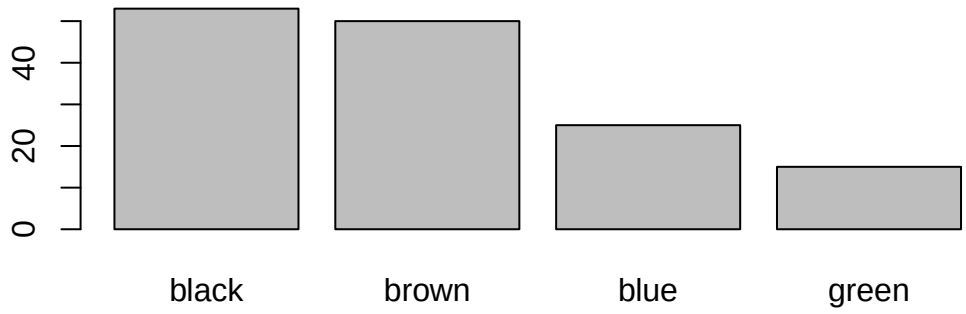


Figure 2.2: Bar chart of brown-haired male eye-colour.

2.3 Classifying basic models by data type

The heart of statistical modelling lies in determining the relationships between different variables. The goals of statistical analysis are either prediction and explanation. For example, one might want to predict a future value of a random variable, called the response variable, given values of other variables, variously called predictor variables, covariates, or explanatory variables. The latter term is more appropriate when thinking of the modelling problem as one of attempting to explain or understand how the response variable relates or is associated with the other variables.

The following table may be useful in organizing your thoughts as to the best form of analysis for given types of data. It is important to remember that this table does not exhaustive of the kinds of statistical analyses that could be undertaken. The ones listed are the most commonly encountered.

response \ covariates	continuous	categorical	both
continuous	regression, correlation	t-test, ANOVA, Wilcoxon, Sign tests	ANCOVA
categorical	logistic	contingency tables	logistic

Regression refers to both simple and multiple regression (which involves more than 1 covariate). ANOVA refers to the analysis of variance, whereby means of different treatment groups are constrasted, depending on the levels or combination of levels from one or more factors. Factors are essentially another way of referring to categorical variables. Block designs are included in this category, and involve a factor which is not of direct interest to the scientist but which is known or believed to have an affect on the response. By including blocking factors in such analyses, more precision (i.e. less uncertainty) can be gained. ANOVA can also be viewed as a type of regression where the covariates are categorical and are made numeric by the binary variable coding described earlier.

ANCOVA is the analysis of covariance, which can be viewed as regression with both continuous and categorical predictors, or as a way of doing ANOVA (i.e. comparing treatment means), accounting for continuous covariates; this is a way of blocking with continuous covariates.

Logistic regression refers to the modelling of the probability distribution of a binary response variable, and the modern view of statistics sees logistic regression as encompassing contingency table analysis. In other words, contingency table analysis can be accomplished by performing logistic regression with categorical covariates.

2.4 Stochastic models

Earlier in this chapter, we saw an example of a numeric data type, the Lake Huron lake levels, that was described as a time series. Specialized models have been developed to account for the dependencies in such data and which lead to more realistic and accurate predictions than could be obtained if this structure is ignored. The simplest time series models essentially amount to regressions of later observations on earlier observations - autoregression or self-regression, and a tool analogous to the correlation, called autocorrelation is often helpful in elucidating the dependence structure in time series data.

Data having categorical labels, as opposed to numeric labels, can also possess dependence structures. One approach to such data is through Markov chain models which specify the conditional probability that a later observation might belong to a certain category, given that an earlier observation was in another given category. A simple example concerns the state of health for an individual through time; such an individual might be healthy, ill or dead at any given time. The probability of entering the death state from the ill state is likely different than what it would be from the healthy state, and an individual will transit back and forth between healthy and illness states many times before entering the death state. Because there is considerably uncertainty in predicting the times of transition from state to state, such a process is referred to as a stochastic process.

Other kinds of stochastic processes include models for Brownian motion, for example, which have been used successfully in analysis of financial data, such as stock returns.

Exercises

1. Construct a histogram plot of the data in the `rivers` data set.[1] Describe the shape of the distribution.

2. Construct a histogram of the rivers data on the log scale.[2] Describe the shape now.

3. Why do you think a bar chart is more appropriate than a pie chart for visualizing categorical data?[3]

[1] This object is built into R and contains the lengths of the 141 longest rivers in North America.
[2] Use `log(rivers)` to obtain the natural logarithm of the river lengths.
[3] Discerning differences in areas and angles is more difficult than discerning differences in heights.

4. The default colour scheme for most plots in R is gray-scale and not colour. What are the reasons for avoiding colour when visualizing data?[4]

5. The `HairEyeColor` data set in R contains sample information on hair and eye colour for males and females. If you type `HairEyeColor`, you will see two contingency tables of hair colour versus eye colour for the two sexes. You can access the blond female eye colour information directly, by typing

```
HairEyeColor[ ,4, 2]
```

   Construct a bar chart for the eye colour of blond females by typing

```
barplot(HairEyeColor[ ,4, 2])
```

   What happens when you omit the '2'?

6. Type `help(InsectSprays)` to find information on this data set. Then construct a histogram of the counts of insects in the various experimental plots by using

```
hist(InsectSprays$count)
```

   Note the shape of the distribution and re-draw the histogram using the `sqrt` function, that is, by applying a square root transformation to the counts beforehand. How does the distribution shape change.

7. Re-do the previous exercise with box plots. Then try

```
boxplot(count ~ spray, data = InsectSprays)
```

   and repeat using the square root transformation of the counts. What is the effect of the square root transformation here?[5]

8. In the previous question, what type of data analysis is recommended?[6]

9. Type `help(airquality)` to find information on the `airquality` data set. Then construct a histogram of `airquality$Ozone`. Repeat using a log-transformation. Which is closer to normality?

10. If you were to model Ozone level as it relates to Wind, what analysis technique is recommended?[7] What if you take temperature into account?[8] What if you add in the `Month` variable?[9]

2.5 T-tests

The goal of these tests, and the related confidence intervals, is to provide information about the mean of a single population, or about the difference in means of two populations. The critical assumption underlying the t-test is independence of the measurements. In other words, if you know the value of one or more of the observations, you cannot predict another observation or group of observations with improved certainty.

We will demonstrate the techniques with examples.

[4] Reproducing plots on hard copy often uses gray-scale, and more importantly, a surprisingly large proportion of the human population is colour-blind.

[5] The variability in the different distributions is better approximated by a constant after applying the square root transformation.

[6] ANOVA

[7] Simple regression.

[8] Multiple regression.

[9] There are choices here, but ANCOVA is a simple option.

2.6 One sample

A soft-drink dispensing machine is supposed to fill paper cups with 250 mL of the liquid, on average. To check whether the machine is working properly, a sample of 10 measurements was taken:

```
##  [1] 251.6 248.0 249.5 251.1 250.0 248.9 251.6 248.9 250.5 249.6
```

We can calculate the mean and standard deviation for this sample using the `mean()` and `sd` functions:

```
mean(volume)

## [1] 249.97

sd(volume)

## [1] 1.220246
```

Clearly, the sample mean is not 250, but the true mean could still be 250, and this result could just be the result of random chance. The t-test helps us answer this question:

```
t.test(volume, mu = 250,  conf.level = .995)

##
##  One Sample t-test
##
## data:  volume
## t = -0.077745, df = 9, p-value = 0.9397
## alternative hypothesis: true mean is not equal to 250
## 99.5 percent confidence interval:
##  248.5462 251.3938
## sample estimates:
## mean of x
##    249.97
```

The p-value is large, telling us that the sample provides no evidence against the hypothesis that the true mean is 250.

2.6.1 Two independent samples

If we have two random samples that are independent of each other, we can still use a t-test to compare the means of the populations from which they were sampled.

The `lengthguesses` data set in the *MPV* package gives a useful example. Here, 44 students were supposed to guess the length of the auditorium they were sitting in, to the nearest meter, and 69 other students were supposed to guess the length to the nearest foot. The guesses have been converted to meters, and the true length of the auditorium is 13.1 meters. The guesses that had been in feet are labelled as `imperial` and the guesses in meters are labelled as `metric`.

```
library(MPV)
with(lengthguesses, t.test(imperial, metric))

##
##  Welch Two Sample t-test
##
## data:  imperial and metric
## t = -2.3126, df = 58.774, p-value = 0.02427
```

```
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -5.0497216 -0.3645571
## sample estimates:
## mean of x mean of y
##  13.31559  16.02273
```

From the output, we see that the guesses in feet were more accurate, on average, than the guesses in meters, and, on the basis of the small p-value for the test (less than 2.5%), we have fairly strong evidence of a difference between the two types of guesses.

2.6.2 Two samples - matched pairs

Darwin's mignonette data compare the heights of cross-fertilized with self-fertilized plants. The plants were paired within pots. The `mignonette` data frame contains two columns of heights: `self` and `cross`.

Let's see what the t-test says:

```
library(DAAG) # the data set is in DAAG package
attach(mignonette) # attaching allows us access to the columns
t.test(self, cross, paired=TRUE)

##
##  Paired t-test
##
## data:  self and cross
## t = -2.1169, df = 23, p-value = 0.0453
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -5.05635317 -0.05823016
## sample estimates:
## mean of the differences
##               -2.557292

detach(mignonette)
```

The p-value is moderately small indicating that there is moderate evidence of a difference. In practice, it would be advisable to collect additional data before making a firm conclusion.

2.7 Simple Regression

The yield (y, in kg/plot) was measured for various salinity concentrations (x, measured in units of electrical conductivity). 18 measurements were recorded in a file called *tomato.txt* whose contents appear below:

The first column contains the salinity concentration levels, and the second column contains the yield measurements.

We read these data into R using the `read.table()` function (or using a menu option in RStudio):

```
tom <- read.table("tomato.txt", header=FALSE)
```

Since there is no header, we should apply some sensible names to the data frame:

```
names(tom) <- c("salinity", "yield")
```

We next construct a scatterplot of the data to look for patterns and outliers as in Figure 2.3.

1.60	59.50
1.60	53.30
1.60	56.80
1.60	63.10
1.60	58.70
3.80	55.20
3.80	59.10
3.80	52.80
3.80	54.50
6.00	51.70
6.00	48.80
6.00	53.90
6.00	49.00
10.20	44.60
10.20	48.50
10.20	41.00
10.20	47.30
10.20	46.10

```
plot(yield ~ salinity, data = tom)
```

We have plotted `yield` against `salinity` since we take it that the response variable is yield and the explanatory variable or predictor variable is salinity. We base this on the fact that the yield was measured at various salinity concentrations that appear to have been set by the experimenter.

The scatterplot gives an indication of a clear downward trend as salinity increases. The trend is also vaguely linear. The suggestion in the graph is that yield could be predicted by salinity. We can investigate this with the `lm()` function:

```
tom.lm <- lm(yield ~ salinity, data = tom)
```

Note that the `model formula` used here, i.e. `yield ~ salinity`, is identical to that used in the `plot()` function. This is often the case: if you can figure out a good way of plotting the data, this often suggests the form of analysis.

We can explore the output from the fitted model using the `summary()` function:

```
summary(tom.lm)

##
## Call:
## lm(formula = yield ~ salinity, data = tom)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.9560 -1.9665  0.1729  1.8255  4.8440
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  60.6700     1.2807  47.372  < 2e-16
## salinity     -1.5088     0.2005  -7.527 1.21e-06
##
## Residual standard error: 2.828 on 16 degrees of freedom
```

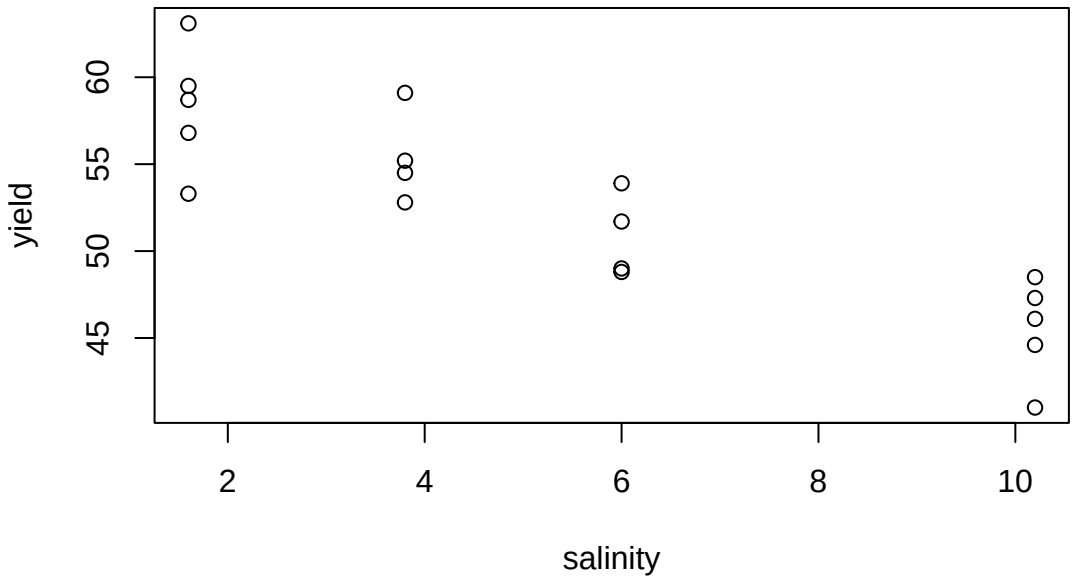


Figure 2.3: Scatterplot of tomato electrical conductivity data.

```
## Multiple R-squared:  0.7798,Adjusted R-squared:  0.766
## F-statistic: 56.65 on 1 and 16 DF,  p-value: 1.212e-06
```

From the output, we can see the estimates of the intercept and slope of the line. Note that the slope estimate, -1.5088 is negative, corresponding to the negative relation between salinity and yield The intercept is 60.67. In fact, we can overlay the scatterplot of the data with the fitted line using the `abline()` function and the output from the `lm()` function:

```
abline(tom.lm)
```

The F-statistic and corresponding p-value ($1.212e-06$, a very very small number) suggest strongly that the slope is nonzero, so the trend in the data is real, provided certain assumptions are satisfied:

1. the relation between salinity and yield is (at least approximately) linear.
2. the measurements are independent of each other, meaning that knowledge of one measurement does not give you information about any other measurement, beyond what you would be able to predict from the line itself.
3. variability in the yield measurements is the same for all salinity levels.

The first and third assumptions are fairly easy to check, and Figure 2.4 helps to do this. A plot of residuals (differences between what the line would predict and the yield measurements) is a clearer way of checking.

The second assumption is difficult to check. It is connected intimately to the manner in which the data have been collected. We will see later that there are some kinds of dependence that can be assessed, but those methods are not applicable here.

From the output, we also see the R^2 and adjusted R^2 values. Although these quantities are often quoted in the scientific literature and used to justify or validate models, they are actually not very useful, and have little to

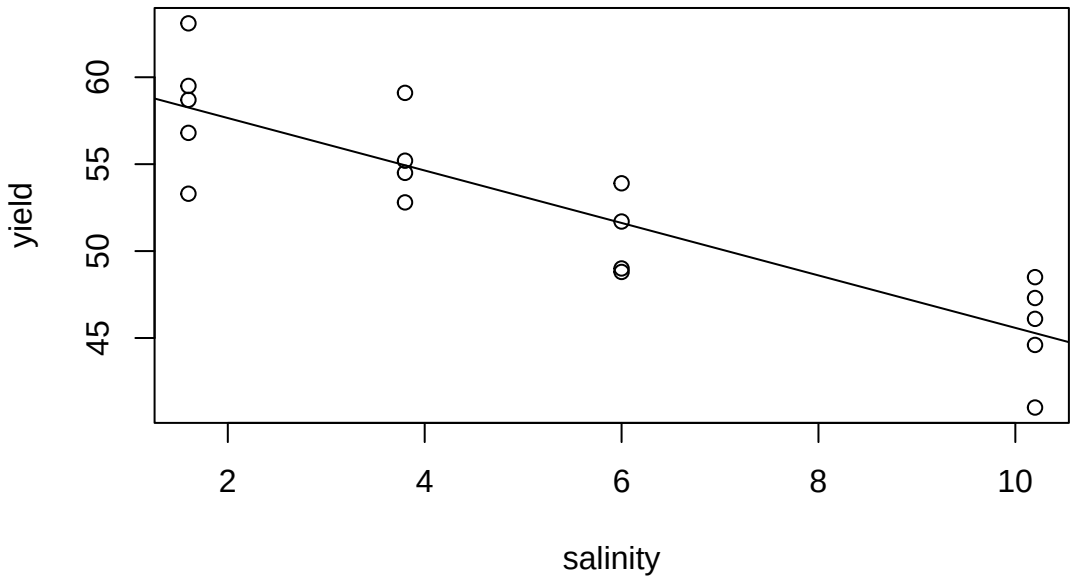


Figure 2.4: Scatterplot of tomato electrical conductivity data with overlaid fitted line.

say about whether a model is valid or not. The proper interpretation of R^2 is as the proportion of variation in the response explained by the model. The coefficient of determination coincides with the square of the Pearson correlation coefficient:

```
cor(tom$salinity, tom$yield)

## [1] -0.8830443
```

By itself, this value tells us that the two variables are negatively correlated, meaning that if one were to plot one of the variables against the other, we would see points scattered about a line with negative slope. In fact, we did just that in Figure 2.4, suggesting that there is limited, if any, information to be gleaned from calculating a correlation coefficient, when the power of a regression analysis is at our disposal. If one really wants to compute a Spearman rank correlation, appropriate for data where a linear trend is not in evidence, one can use

```
cor(tom$salinity, tom$yield, method = "spearman")

## [1] -0.8939601
```

This also is of limited additional use.

Exercises

1. Consider the data on the model car that was released from various points on a ramp and the distance traveled was measured. The data frame is called `modelcars`, and it consists of two columns, `distance.traveled` and `starting.point`.

```
library(DAAG) # the package containing the model car data set
mcar.lm <- lm(distance.traveled ~ starting.point, data = modelcars)
summary(mcar.lm)$coefficients

##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)    8.083333  1.0779514  7.498792 2.065661e-05
## starting.point 2.013889  0.1312041 15.349288 2.801914e-08
```

Identify the slope and intercept of the line relating distance traveled to starting point.[10] Is there strong evidence of a nonzero slope to this line?[11] Is the slope positive or negative?[12]

2. Write down the two lines of R code which would produce the graph in Figure 2.5. [13]

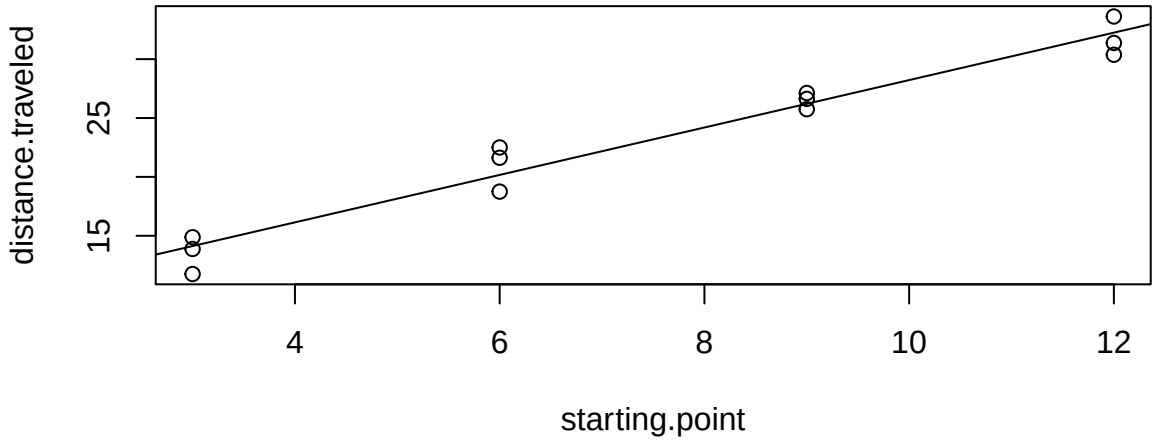


Figure 2.5: Distance travelled by a model car launched from a ramp at various starting points.

```
plot(distance.traveled ~ starting.point,
     data = modelcars)
abline(mcar.lm)
```

3. Analyze the `airquality` data to determine the relationship between ozone and wind, by finding the slope and intercept of the linear model. Then plot the line on a scatterplot of the data.

4. Repeat the above analysis using a square root transformation on the Ozone variable. Is this a better way of modelling the data?[14]

2.8 ANOVA

2.8.1 One factor

The `chickwts` data frame contains measurements of the weights of chicks who have been randomly assigned to groups, each of which has been given a different type of feed. It is of interest to know whether the different feed types lead to systematic differences in weight. In other words, does mean weight depend on the type of feed.

[10] intercept: 8.083; slope: 2.014
[11] Yes, the p-value is $2.802e-08$ which is extremely small.
[12] Positive.
[13] plot(distance.traveled starting.point, data = modelcars); abline(mcar.lm)
[14] Yes, the square root transformation reduces some of the nonlinearity which is apparent when working with raw Ozone data.

We refer to feed type as a factor having different levels representing the particular kinds of feed, e.g. linseed, horsebean, and so on.

Side-by-side boxplots, as displayed in Figure 2.6, are a useful way to visualize these data.

```
plot(weight ~ feed, data = chickwts, cex.axis=.75) # feed must be a factor
```

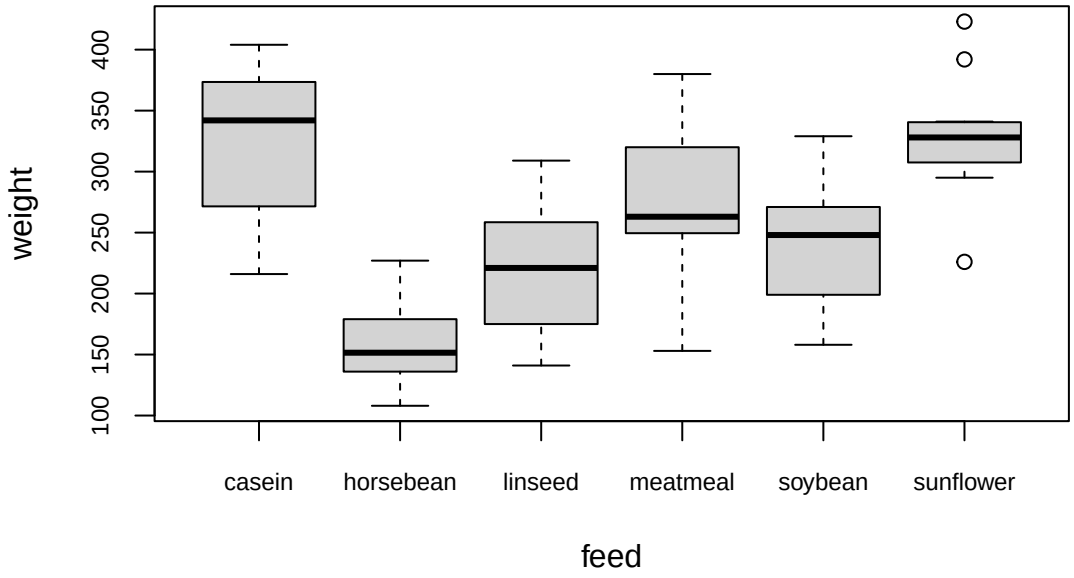


Figure 2.6: Box-plots of chick weight samples for different types of feed.

From the graph, it seems that horsebean leads to lower weights than some of the other feed types. It is hard to tell for sure if there is variability between treatments because of the variability within treatments, that is noise due to unmeasured factors.

We can test whether there is a difference in the mean weights statistically with the analysis of variance (ANOVA). A general purpose procedure is as follows:

```
chick.lm <- lm(weight ~ feed, data = chickwts)
anova(chick.lm)

## Analysis of Variance Table
##
## Response: weight
##           Df Sum Sq Mean Sq F value    Pr(>F)
## feed       5 231129   46226  15.365 5.936e-10
## Residuals 65 195556    3009
```

The test statistic compares the variability in the averages with the variability in the noise through an F-statistic. A p-value is computed which gives the strength of evidence against the null hypothesis, that is the hypothesis that there is no difference in the means. A small p-value – and in this case, it is very small – indicates strong evidence against the null hypothesis, in favour of the alternative that there is a difference.

2.8.2 *Two or more factors*[15]

Information on gas mileage for a number of cars is available in `table.b3` of the *MPV* package. Although not from a designed experiment, we will analyze this observational data as if it were. Our objective is to see if mean gas mileage y depends on either or both of carburetor barrels (x_6, viewed as a categorical variable) and type of transmission x_{11}.

```
library(MPV)
b3.lm <- lm(y ~ factor(x6>1)*x11, data = table.b3)
anova(b3.lm)

## Analysis of Variance Table
##
## Response: y
##                     Df Sum Sq Mean Sq F value    Pr(>F)
## factor(x6 > 1)       1   0.30    0.30  0.0176   0.89530
## x11                  1 688.99  688.99 40.8450 6.433e-07
## factor(x6 > 1):x11   1  75.95   75.95  4.5023   0.04283
## Residuals           28 472.31   16.87
```

A number of points need to be made. First, `x6` is recorded in the data frame as if it is continuous (and it could be treated as such), so in order to treat it as categorical, we use the `factor()` function. We have also coded the variable to have more than 1 barrel or to have 1 barrel.

The second point is the use of the use of $\star$ which forces the model to include both of the factors as well as interactions between the factors. Interaction effects can play an important role in modelling, since they reflect situations where, for example, changing the level of one factor, might have different effects, depending on the level of the other factor. In this case, increasing number of carburetor barrels might affect gas mileage differently for automatic than for manual transmission. The output can help us determine if this is actually the case.

In fact, the p-value for the interaction effect is just under 5%, so there is moderate evidence of an interacting effect between these two variables. The effect of the number of carburetor barrels on gas mileage could be different for manual and automatic transmissions. If this were part of an actual research study, it is critical that this result would be reproduced in a designed experiment. Publishing a marginally significant result such as we obtained here, based on a small sample coming from an observational study would be irresponsible and reckless. Unfortunately, this analysis mirrors too closely for a lot of what passes as scientific research in the published literature.

Exercises

1. Consider the data in `PlantGrowth` and conduct an analysis of variance to determine if the mean dried yield weight of the plants under study differs depending on whether the plants were grown under control conditions or under either of two different treatment conditions.

   Visualize the data with side-by-side boxplots.

2.9 Multiple Regression

The data frame `table.b4` in the *MPV* library contains the following columns:

```
y sale price of the house (in thousands of dollars)
x1 taxes (in thousands of dollars)
x2 number of baths
x3 lot size (in thousands of square feet)
x4 living space (in thousands of square feet)
x5 number of garage stalls
```

[15] This topic may be omitted.

```
x6 number of rooms
x7 number of bedrooms
x8 age of the home (in years)
x9 number of fireplaces
```

There are 24 observations on these variables in the data frame. A natural question to ask is whether any or all of the given variables or covariates could be used to predict the sale price of a house.

The multiple regression approach is to consider a linear model of the form

$$y = \beta_0 + \sum_{j=1}^{9} \beta_j x_j + \varepsilon.$$

If the β coefficients were known, then we could predict house price y, to within the unknown noise value ε, for a new house in the same area (and era) from which the data were sampled, provided the information on taxes, number of baths, and so on. If, for example, the ε term is modelled as a normal random variable with mean 0 and variance σ^2, we could provide an interval which would contain the true price of the house with a given probability.

The elements of ε are assumed to be uncorrelated random variables with mean 0 and common variance σ^2. The mean or expected value of y for the given values of the covariates is then

$$E[y] = \beta_0 + \sum_{j=1}^{9} \beta_j x_j.$$

(The "E" notation stands for "Expected Value".)

2.9.1 *Fitting the model*

The `lm()` function will take care of the coefficient estimation, variance estimation, t and F and p-value calculations in one function call. For example, if we want to relate house price, y to x_1, x_3 and x_6, use

```
house.lm <- lm(y ~ x1 + x3 + x6, data=table.b4)
```

We can view the output from this, using the `summary` function as in

```
summary(house.lm)
```

or to see the coefficient estimates and their statistical properties only, use

```
summary(house.lm)$coefficients
```

If we include all of the covariates, we can use the dot notation in the model formula:

```
house.lm <- lm(y ~ ., data=table.b4)
```

```
summary(house.lm)$coefficients
```

```
##                Estimate Std. Error    t value   Pr(>|t|)
## (Intercept) 14.92764759  5.9128516  2.5246106 0.02428304
## x1           1.92472156  1.0299013  1.8688408 0.08271059
## x2           7.00053420  4.3003717  1.6278905 0.12583613
## x3           0.14917793  0.4903874  0.3042043 0.76544692
## x4           2.72280790  4.3595535  0.6245612 0.54230434
## x5           2.00668402  1.3735086  1.4609912 0.16609649
## x6          -0.41012376  2.3785444 -0.1724264 0.86557016
## x7          -1.40323530  3.3955419 -0.4132581 0.68567761
## x8          -0.03714908  0.0667199 -0.5567916 0.58646103
## x9           1.55944663  1.9374959  0.8048774 0.43434718
```

The output above lists a number of things. Our focus is principally on the `Estimate` column, since that gives us the estimates of the coefficients β. The intercept is 14.93 and the coefficient of x_1 is 1.92, and so on.

Together with these estimates are estimates of the standard errors. These provide an assessment of the amount of uncertainty is associated with the corresponding coefficient estimate. Clearly, the estimate of β_6 has a relatively large degree of uncertainty associated with it, since the standard error is much larger than the absolute value of the estimate itself.

This highlights an important problem when applying multiple regression techniques: over-fitting. When using a limited amount of data to estimate a large number of parameters in this case, 10, the degree of uncertainty in the estimates rises quickly. It is often better to carefully decide which covariates to include in a model based on other considerations. Use any known science or other information to help make these choices. For example, it might be known that taxes and living space are highly predictive of sale price. In that case, focus on those variables immediately in order to more precisely estimate their coefficients.

```
house.lm <- lm(y ~ x1 + x4, data=table.b4)
summary(house.lm)$coefficients

##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 11.544687  3.1747354 3.6364251 1.544365e-03
## x1           2.919510  0.5759885 5.0686947 5.097155e-05
## x4           3.157449  3.2983334 0.9572861 3.493147e-01
```

Notice how the standard error estimates for the coefficients of x_1 and x_4 are less than before, although the standard error for the living space variable still exceeds the absolute value of the coefficient, so there is considerable uncertainty left there.

2.9.2 *Estimating and predicting*

The model can now be used to estimate the expected house price for houses with x_1 taxes and x_4 amount of living space using the formula

$$\widehat{y} = 11.5 + 2.92x_1 + 3.15x_4.$$

This can be accomplished in R using the `predict()` function. For instance, suppose we want to estimate the mean sale price for homes with $2000 taxes and 3000 square feet of living area. Use

```
predict(house.lm, newdata = data.frame(x1 = 2, x4 = 3))

##        1
## 26.85605
```

The mean price for such a home is $25856. This estimate highlights an important point: this data set is old and it applies to a particular location. In order to understand the housing market in a particular location and time, it is necessary to use the relevant data.

Note also that the `predict()` function has been used. It can be used both for estimating the mean price of a house or predicting the price of a specific house. Prediction uncertainty is usually much larger than estimation uncertainty; both are incorporated in the `predict()` function. Use `interval = "confidence"` for estimation and `interval = "predict"` for prediction. For example,

```
predict(house.lm, newdata = data.frame(x1 = 2, x4 = 3), interval = "predict")

##        fit      lwr      upr
## 1 26.85605 10.23286 43.47925
```

This says that with 95% probability, the house we are looking at with taxes of $2000 and 3000 square feet of living area is priced between $10232 and $43479.

```
predict(house.lm, newdata = data.frame(x1 = 2, x4 = 3), interval = "confidence")
```

```
##        fit      lwr      upr
## 1 26.85605 11.42015 42.29196
```

This says that with 95% confidence, we can say that the mean price of houses with taxes of $2000 and 3000 square feet of living area is between $11420 and $42292.

2.9.3 *Assessing the model*

In the past, the term model validation was used when describing the process of deciding whether a model was appropriate or not. This incorrectly conveys the sense that there is a correct model; it is now recognized that all models are incorrect at some level, but some are more useful for certain purposes than others might be. Thus, the term model assessment is now favoured, since it conveys a sense of checking the appropriateness of a given model as opposed to checking its validity.

Although there are a number of statistics that are often (ab)-used to do this assessment, a graphical approach is usually the best way to understand whether there are substantive problems with a given model. In particular, graphs of residuals are the best way to decide if a model is failing severely. A residual is the difference between the observed response value and the value predicted by the model. As such, residuals are effectively predictors of the noise term ε in the model.

Since the noise term is assumed to have mean 0, constant variance and to have no internal correlations, we can often simply look at a graph of the residuals plotted against observation number, fitted value, or a predictor variable to look for patterns. Clear patterns are a sign of severe model failure. Figure 2.7 displays the residuals plotted against the fitted values, with a smooth fitted overlaid red curve which can guide the eye to any systematic patterns. In this case, the curve is not substantially different from a flat line, which would be ideal. There does not seem to be a clear pattern in the residuals in this case.

```
plot(house.lm, which = 1)
```

Another important plot, such as in Figure 2.8, concerns the influence of individual data points on the model fit. Large values of Cook's distance are suggestive of difficulties which should be remedied. For assistance with such problems, it is probably best to consult your local statisticians for help. The values seen in the current case are not worrisome.

```
plot(house.lm, which = 4)
```

2.9.4 *Significance of regression*

The F-test for significance of regression gives us a way of deciding whether any of the regression coefficients should be nonzero. In other words, the coefficients (apart from the intercept) are assumed to be 0's under the null hypothesis and there must be at least one nonzero coefficient if the alternative hypothesis is true. An F distribution is used to conduct the test. The p-value based on the test gives us the strength of evidence supporting the case that at least one regression coefficient is nonzero, where, as usual, small values indicate more evidence than large values would.

We can test whether the coefficients of x_1 and x_4 are both 0 in a variety of ways, including by looking at all output from `summary(house.lm)`. In order to see more clearly what is happening, you can use the `anova()` function to decide between the null model (one with only an intercept) and the model we have already fit:

```
house0.lm <- lm(y ~ 1, data = table.b4) # intercept only model
anova(house.lm, house0.lm) # test significance of regression
```

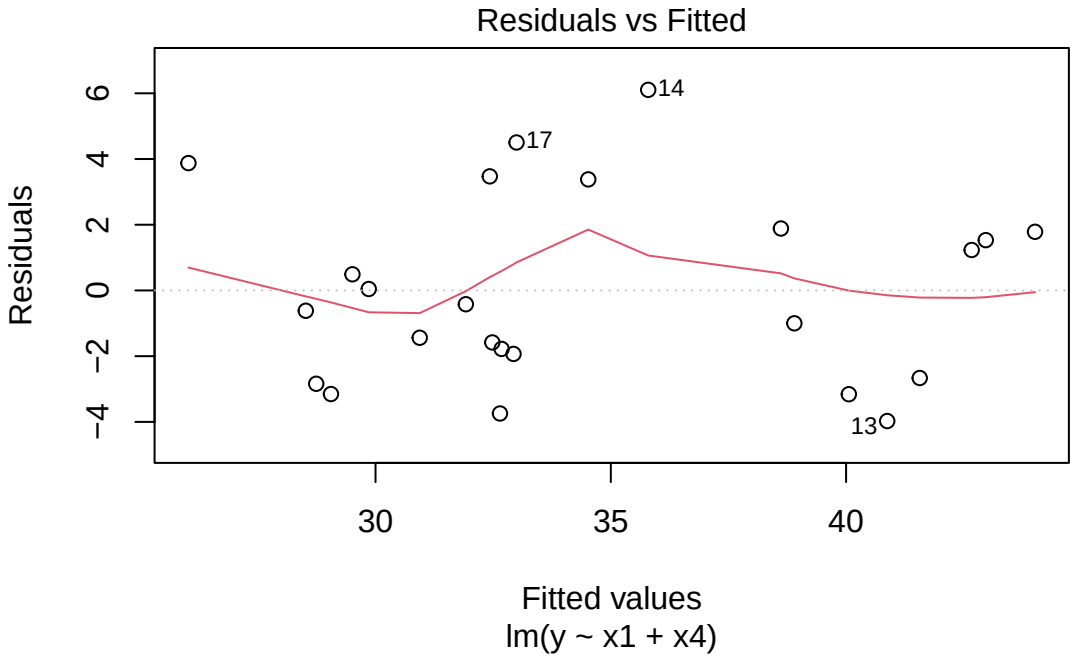


Figure 2.7: Plot of residuals for house price model against fitted values.

```
## Analysis of Variance Table
##
## Model 1: y ~ x1 + x4
## Model 2: y ~ 1
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1     21 184.83
## 2     23 829.05 -2   -644.22 36.599 1.432e-07
```

The p-value is small indicating that at least one of the coefficients is nonzero. The nice thing about this approach is that we can use it to decide if more variables should be added to the model. For example, let's see if there are any other variables to add, after adding x_1 and x_2:

```
houseAll.lm <- lm(y ~ ., data = table.b4) # fit with all variables
anova(houseAll.lm, house.lm) # compare All- and two-variable models

## Analysis of Variance Table
##
## Model 1: y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
## Model 2: y ~ x1 + x4
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     14 121.75
## 2     21 184.82 -7   -63.077 1.0362 0.4496
```

This analysis tells us that once we have taken taxes and living area into account, there is no point in adding additional variables into the model. There is no evidence to suggest that the additional coefficients are nonzero.

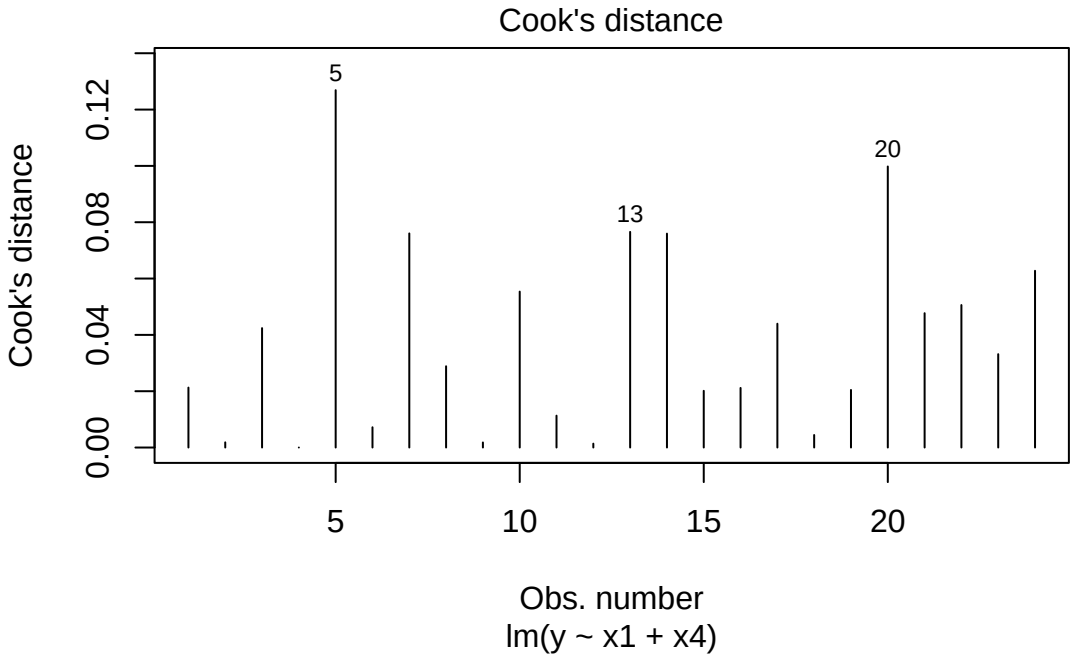


Figure 2.8: Influence diagnostic plot for house price models.

Exercises

1. Consider the gas mileage data in `table.b3` of the *MPV* package.

   (a) Fit a multiple regression model to estimate mean gas mileage y for cars with x_7 number of transmission speeds and having weight x_{10}.

   (b) Assess the model using the residual plot.

   (c) Use the model to estimate mean gas mileage for cars having weight 5000 pounds and 4 transmission speeds. Use a 95% confidence interval.

   (d) Use the F-test for signifiance of regression to decide if any of the coefficients for your fitted model are nonzero.

   (e) Use another F-test to decide if variables x_1, x_2, x_4 or x_5 should be added into the model you have already developed.

2.10 ANCOVA

[16]

The analysis of covariance (ANCOVA) allows us to model continuous responses as linear functions of continuous and categorical covariates. In this way, it can be viewed as a relatively straightforward extension of multiple regression. It can also be viewed as an extension of ANOVA whereby there is a blocking factor which is continuously measured. For a categorical covariate with two levels, there would be two lines in the regression: parallel if there is no interaction effect; two different slopes if there is an interaction effect.

The `ToothGrowth` data frame in R concerns the length of odontoblasts, cells connected with the growth of teeth, in a sample of 60 guinea pigs. One of three dose levels of vitamin C were supplied to the guinea pigs in one

[16] This topic may be omitted.

of two forms: `supp = VC` refers to ascorbic acid and `supp = OJ` refers to orange juice. Figure 2.9 displays the data, using the `xyplot()` function from the lattice package (Sarkar, 2008).

```
library(lattice)
xyplot(len ~ sqrt(dose)|supp, data = ToothGrowth)
```

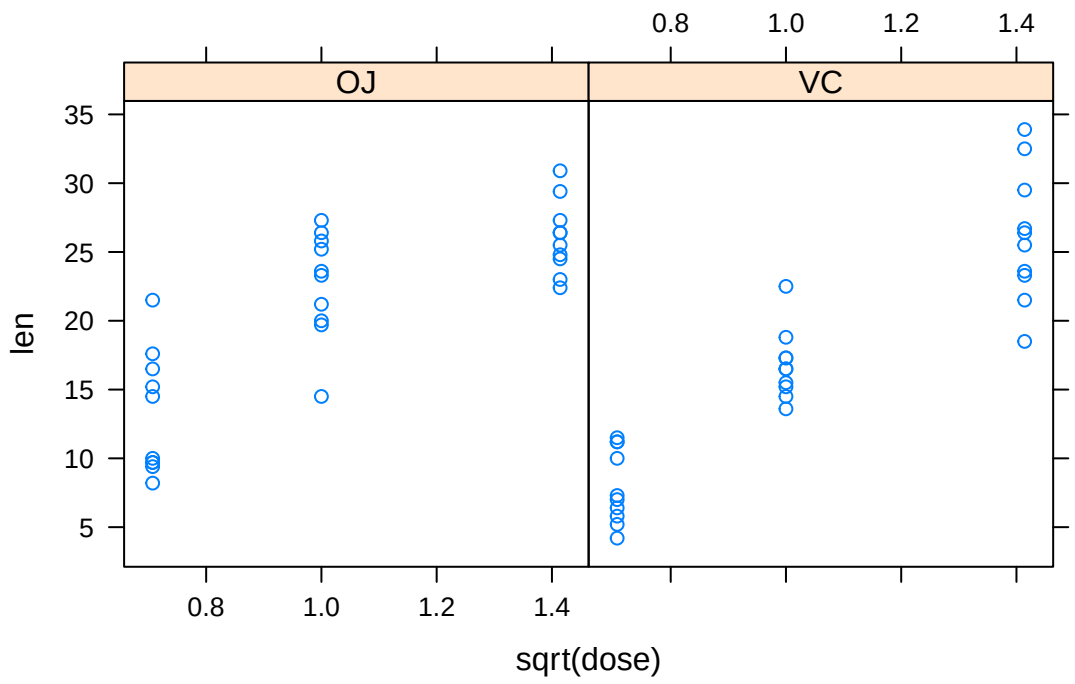


Figure 2.9: Tooth growth data: length of tooth versus square root of vitamin C dose, for each of the two treatment methods.

The figure shows that there are possibly two different lines relating length to vitamin C dose; it is possible that there is a treatment effect.

We use `lm()` to check this, first by allowing for two intercepts and two slopes:

```
TG.lm <- lm(len  ~ sqrt(dose)*supp, data = ToothGrowth)
summary(TG.lm)

##
## Call:
## lm(formula = len ~ sqrt(dose) * supp, data = ToothGrowth)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.9867 -2.6768 -0.1716  2.7380  7.4133
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.478      2.628   0.943  0.34976
```

```
## sqrt(dose)           17.479      2.433   7.185 1.71e-09
## suppVC              -12.024      3.716  -3.236  0.00204
## sqrt(dose):suppVC     8.000      3.441   2.325  0.02370
##
## Residual standard error: 3.865 on 56 degrees of freedom
## Multiple R-squared:  0.7576,Adjusted R-squared:  0.7446
## F-statistic: 58.35 on 3 and 56 DF,  p-value: < 2.2e-16
```

Looking at the interaction between the square root of dose and `supp`, we see a fairly small p-value which is highly suggestive of different slopes. There is no reason to consider the model without different slopes (which would have been obtained by replacing the $\star$ with $+$). The value 8 indicates that for `VC`, the slope is 8 units higher than for `OJ`: $17.48 + 8 = 25.48$. The intercept for `VC` is 12.02 units lower: $2.478 - 12.024 = -9.546$.

We can graphically summarize the data with a simple scatterplot with the lines overlaid:

```
plot(len ~ sqrt(dose), pch = as.numeric(supp), data = ToothGrowth)
abline(2.478, 17.479) # OJ line
abline(2.478-12.024, 17.479 + 8, lty=2) # VC line, dashed
```

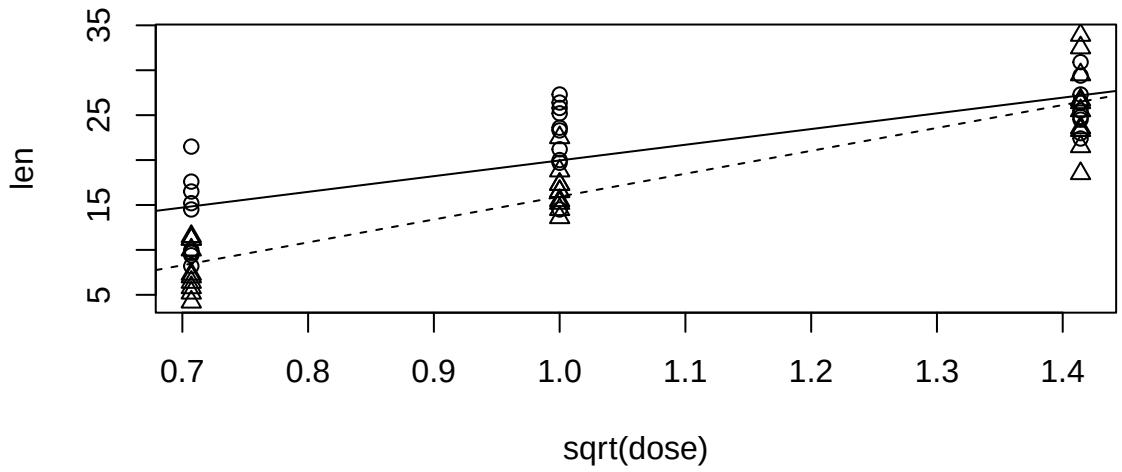


Figure 2.10: Tooth growth data: length of tooth versus square root of vitamin C dose, for each of the two treatment methods. The solid line corresponds to the orange juice treatment, and the dashed line corresponds to the ascorbic acid treatment

Exercises

1. Fit the model with two parallel lines to the tooth growth data. What are the intercepts for the VC and OJ lines? What is the slope? Plot the data with the two lines overlaid.

2. Consider the data in `airquality` which relate to Ozone levels in New York. Construct a model which relates Ozone level to temperature and wind, taking Month into account, as a factor. Is there evidence that Month should be included in the model? Is there an interaction between Month and wind? between Month and temperature?

2.11 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 2.1 *Consider the data in* `table.b3` *of the MPV package. 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 2.11, is the result of executing the following code.

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

Figure 2.11: Default regression tree for car gas mileage data.

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 often be achieved.

Example 2.2 *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 2.12 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")
```

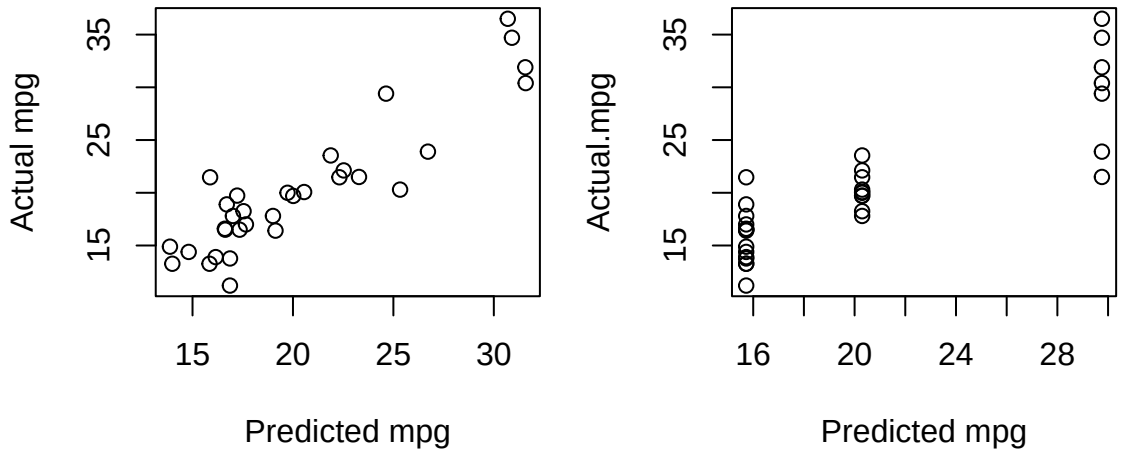


Figure 2.12: Random forest (left panel) and tree-based (right panel) predictions of car gas mileage.

2.12 Logistic Regression

[17]

2.12.1 Modelling binary responses

The data in `p13.1` in the *MPV* package describes successes and failures of surface-to-air missiles as they relate to target speed. The data are plotted in Figure 2.13, with successes on the vertical axis being represented by a '1' and failures being represented by a '0'.

Such binary data are not nearly normally distributed, so it the efficacy of least-squares becomes very questionable here. In this section, we indicate what could and should not be done with least-squares for such data.

```
library(MPV)
plot(p13.1, xlab = "target speed", ylab = "success/failure")
```

The first observation to make is that fitting a straight line to such data makes no sense, since the plotted points do not at all scatter about such a line. Furthermore, if such a line were to be fit to the data, it would necessarily

[17] This topic may be omitted.

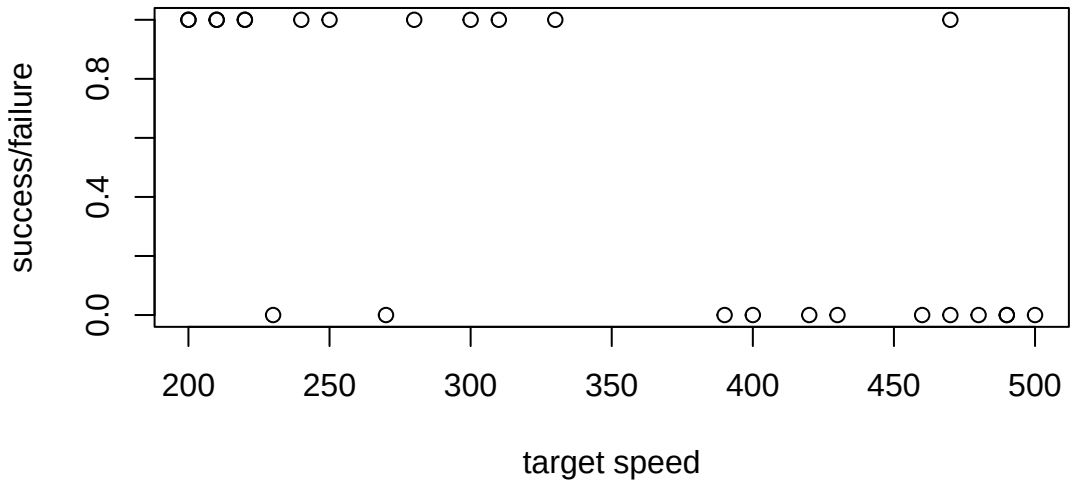


Figure 2.13: Surface-to-air missile successes (1) and failures (0) as they relate to target speed (in knots).

take values outside the interval $[0, 1]$ on subsets of the domain; interpretation of such values would be difficult. In fact, the preferred interpretation of output arising from the fitting of models to such data is that of probability. That is, useful models can provide answers to questions such as, "What is the probability of success at a given target speed?" Since probabilities must lie within the interval $[0, 1]$, we must consider models based on nonlinear functions.

There are many functions which have values in $[0, 1]$. For example, the absolute value of the sine function is a candidate. Such a function might be appropriate if there were oscillatory or periodic behaviour to be modelled, but often, the desired model behaviour is monotonic (either increasing or decreasing). For the current example, we might reasonably believe that the probability of success decreases as target speed increases.

Perhaps the most popular function for this purpose is the logistic function

$$p(x) = \frac{e^x}{e^x + 1}.$$

The function is sketched in Figure 2.14.

```
curve(exp(x)/(1 + exp(x)), from = -3, to = 3, ylab="p(x)")
```

A bit of algebra allows us to express x in terms of p, yielding the logit function:

$$\ell(p) = \log\left(\frac{p}{1-p}\right).$$

While p is restricted to take values between 0 and 1, the logit function can take any possible value, so relating the logit function to a straight line or other linear combination is a possibility. For example,

$$\ell(p(x)) = \beta_0 + \beta_1 x$$

which means that we can express the probability of an event in terms of a covariate x, using a linear function, but the probability is related to the linear function through the logit. This kind of model where a linear function of

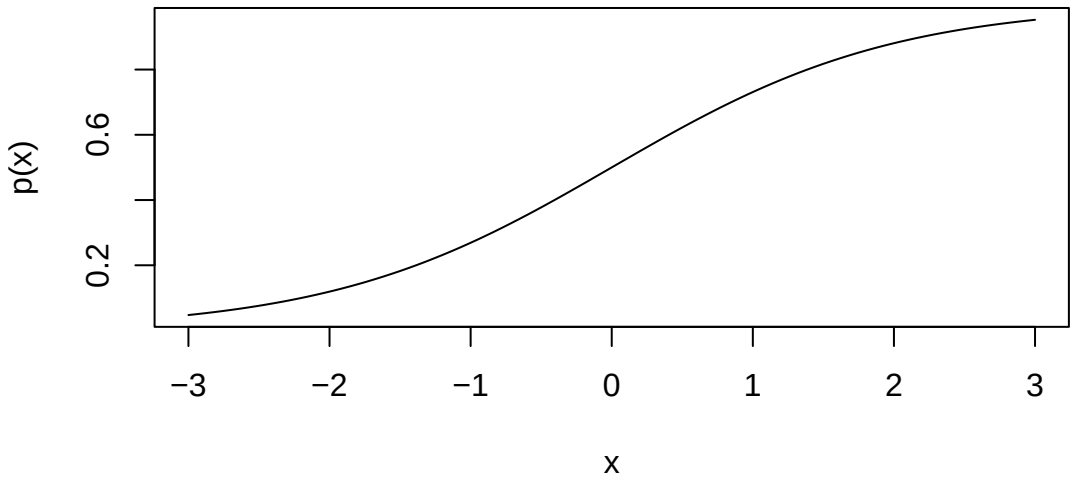


Figure 2.14: The logistic function.

the covariate(s) is related to a function of the expected response is called a generalized linear model. The logit is an example of a link function, since it links the expected response, in this case the probability $p(x)$ to the linear function of the covariate(s). Other link functions that are popular are the probit, and the complementary log-log. The probit is the inverse of the normal probability distribution function. All of these alternatives are available for use in the `glm()` function through the `binomial()` family function.

To fit the logistic regression model to the missile success data, try

```
p13.glm <- glm(y ~ x, data = p13.1, family = binomial)
summary(p13.glm)

##
## Call:
## glm(formula = y ~ x, family = binomial, data = p13.1)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.062  -0.487   0.392   0.548   2.168
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.07088    2.10900    2.88   0.0040
## x           -0.01770    0.00608   -2.91   0.0036
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.617  on 24  degrees of freedom
## Residual deviance: 20.364  on 23  degrees of freedom
## AIC: 24.36
```

```
##
## Number of Fisher Scoring iterations: 4
```

Note that we did not specify the link function; the default choice with the binomial family is the logit.

The Coefficient part of the output tells us that the logit of the probability of success as a linear function of target speed has intercept 6.07 and slope -.0177. Standard error estimates for these parameter estimates are supplied and indicate, in particular, that the slope is clearly negative.

The line itself is not as interesting as the estimated logistic curve which is plotted in Figure 2.15 together with the original data. The curve can now be used to read off specific probabilities of success at the various speeds. Note that in order to obtain the curve, we have used the `predict()` function with `type = "response"`; without specifying `type`, the default is to use the predictions on the linear scale.

```
plot(p13.1, xlab = "target speed", ylab = "success/failure")
newspeeds <- 200:500 # speeds at which we can predict using the fitted model
lines(newspeeds, predict(p13.glm, newdata=data.frame(x = newspeeds),
     type = "response"))
```

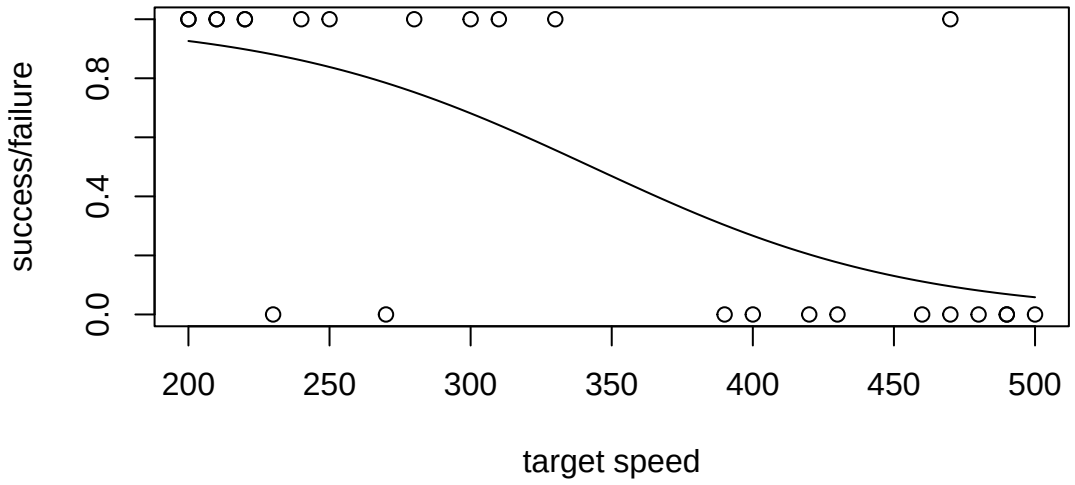


Figure 2.15: Surface-to-air missile successes (1) and failures (0) as they relate to target speed (in knots) with overlaid logistic curve.

2.12.2 *Model adequacy and checking*

Other features of the `glm()` output should be discussed. The dispersion parameter has been taken to be 1. We are assuming that there is no clustering in the data which would have possibly led to overdispersion: the case where the variance exceeds what would be expected under a binomial model. If there is a belief that clustering is occurring (not likely in this example), the `quasibinomial` family should be used instead.

The null deviance refers to a quantity that is calculated for a model that does not include the covariate, in this case `speed`. You can view it and the residual deviance as a generalization of the notion of sum of squares. The

residual deviance is calculated for the model which includes the covariate and is considerably smaller than the null deviance, suggesting that the covariate is useful as a predictor.

Residuals can also be plotted, but with binary data, it is difficult to assess the patterns seen on such graphs.

Exercises

1. Estimate the logit of the probability of missile success at a speed of 400 knots. Calculate the probability of missile success. (For this, you can either use the logistic formula, or the `predict()` function in R, using the correct `type`.)

2. The `p13.2` data frame in the *MPV* package has 20 observations on home ownership as it relates to family income. Fit a logistic regression model to the data and use the output to

   (a) identify the logit of the probability of home ownership as a linear function of family income.

   (b) determine if the logistic model is reasonable.

   (c) estimate the probability that a family with an income of $40000 owns their home.

3

Simulation: Generating and Testing Pseudorandom Numbers

Simulation of realistic scenarios usually requires the incorporation of uncertain or unpredictable elements. Generation of sequences of independent random variables is thus required. This chapter describes how to simulate random variables on a computer. We will describe deterministic 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, hopefully, a good approximation to truly random numbers. Knuth (1997) offers a comprehensive treatment of pseudorandom number generation and testing which remains authoritative.

3.1 Sequential generation of pseudorandom numbers

The most common techniques for generating pseudorandom variables 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.

3.1.1 Linearity is highly predictable

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

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

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 3.1 *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 not a good characteristic of a random number simulator.

3.1.2 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 3.2 *An example of a nonlinear function is the cosine function. We start with* $x_0 = 2$ *and generate 12 successive values from*

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

```
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 3.3 *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 3.1, 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)
```

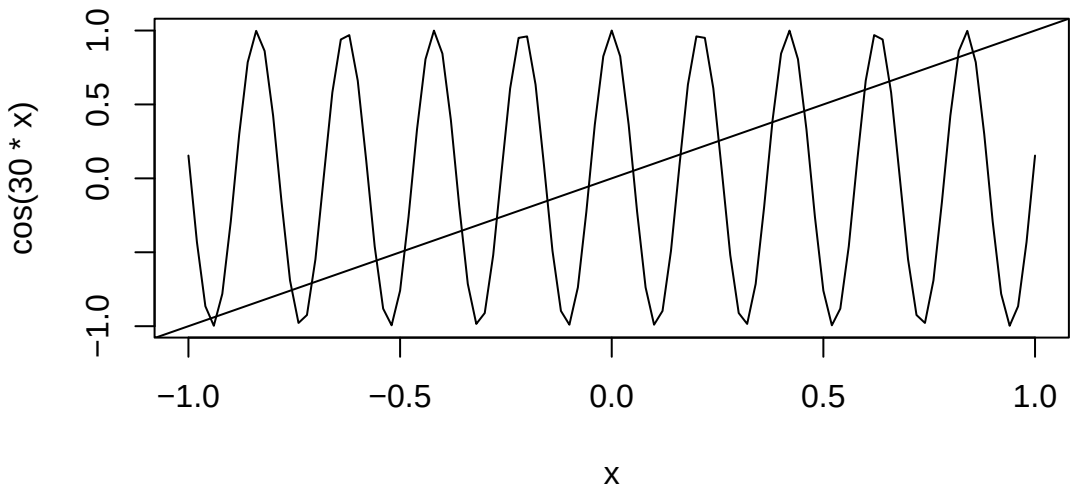


Figure 3.1: A mapping with more, and less stable, fixed points.

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)
```

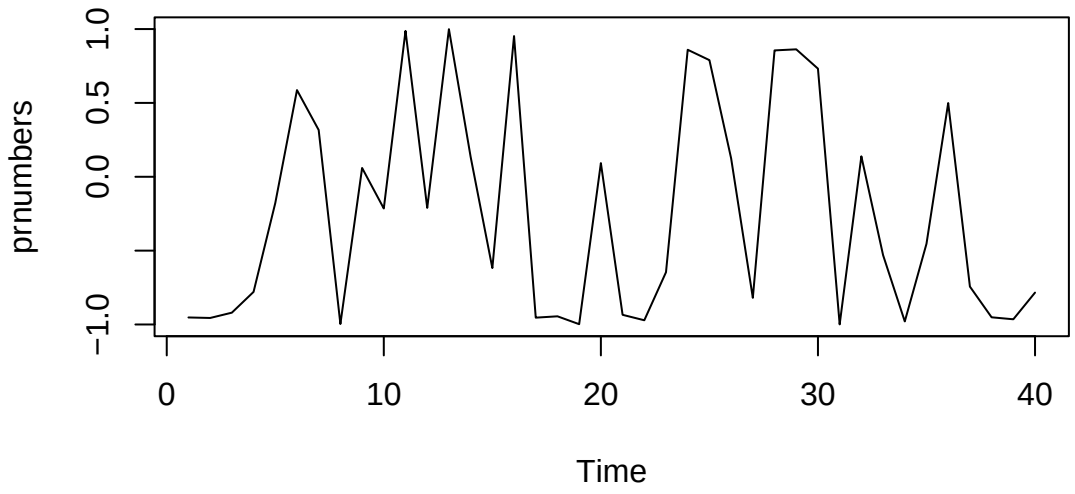


Figure 3.2: Trace plot of first 40 numbers sequentially generated from the cosine map, $\cos(30x)$.

The numbers produced by this function are certainly less predictable than before, as can be seen in the trace plot pictured in Figure 3.2.

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

Example 3.4 *Consider the function* $f(x) = 32678x \mod 33271$*:*

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

Its graph is plotted in Figure 3.3.

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

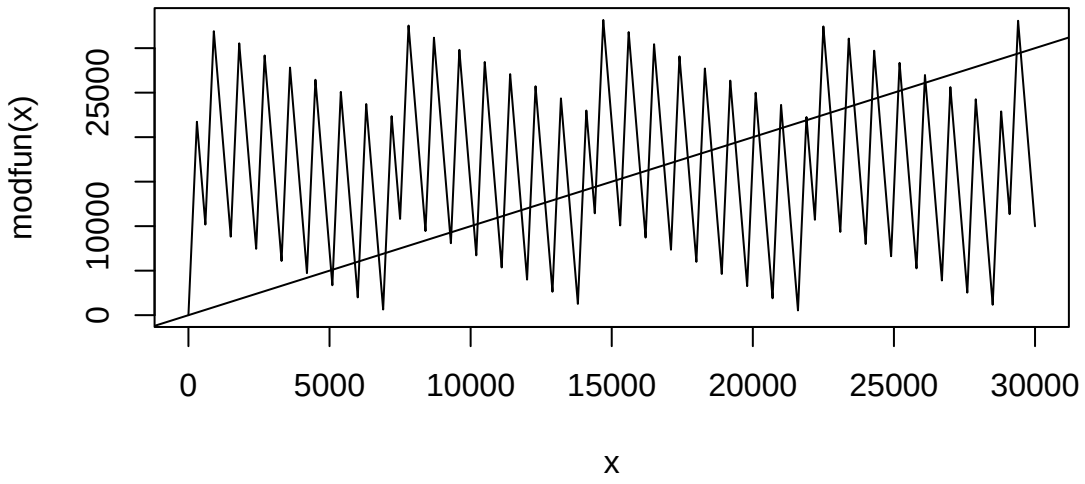


Figure 3.3: An example of a nonlinear function with jumps.

```
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 3.4, showing a much less predictable pattern than any we have seen so far.

3.1.3 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.[1] 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 $n = 1, 2, \ldots$,

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

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

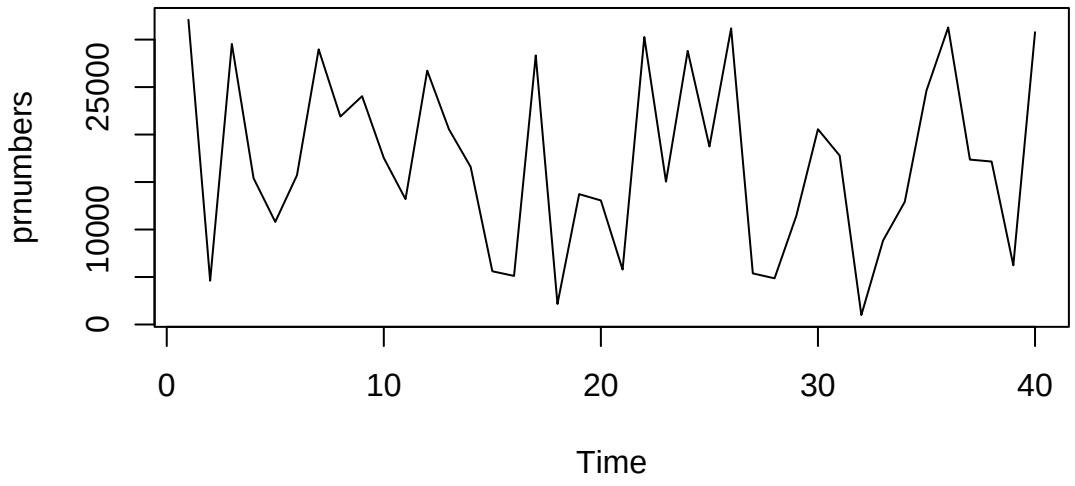


Figure 3.4: Trace plot of the first 40 numbers coming from the nonlinear function with jumps.

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

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

$$u_n = x_n/32$$

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

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

$$u_n = x_n / 30269$$

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 pseudorandom 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.

3.1.4 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
```

3.2 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.

3.2.1 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 3.7 *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)
```

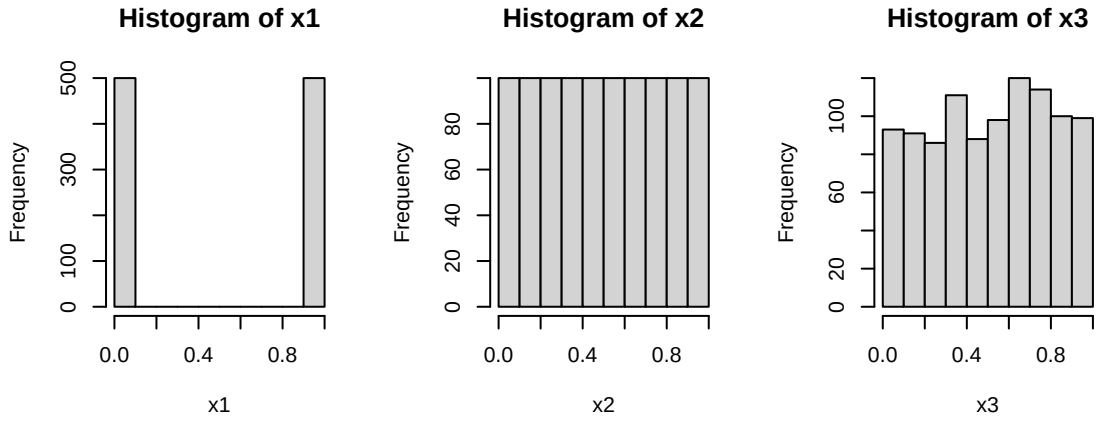


Figure 3.5: Histograms of three sets of pseudorandom numbers.

Figure 3.5 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.*

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.

3.2.2 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 3.8 *In the first example, we see points lying on clearly separated lines in the lag plot displayed in Figure 3.6. This generator is very poor.*

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

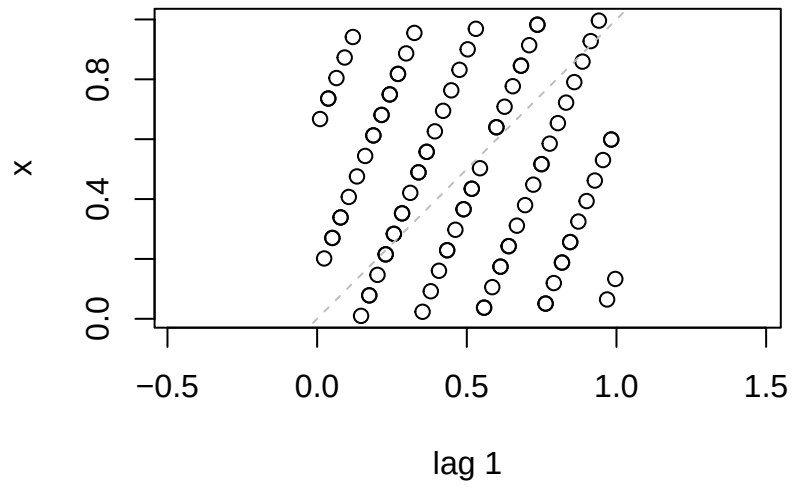


Figure 3.6: Lag plot for a pseudorandom number generator with bad properties.

By changing the a parameter, we get the result plotted in Figure 3.7.

```
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 3.8 confirms this.

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

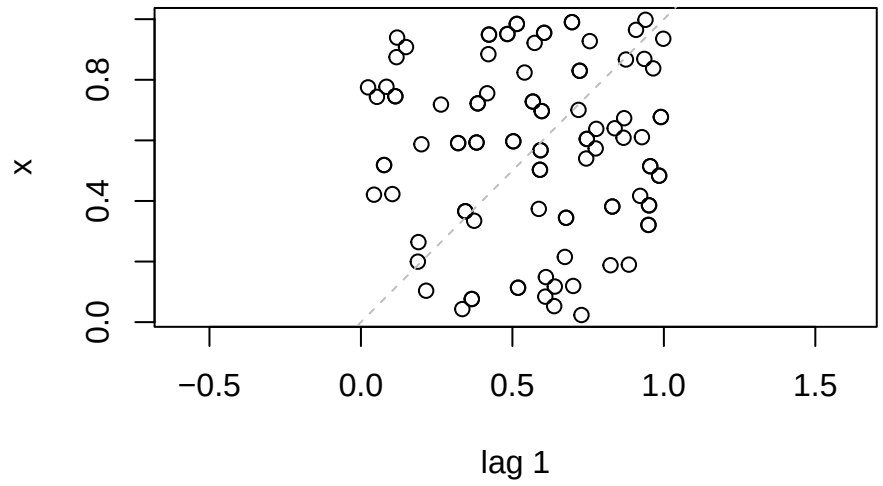


Figure 3.7: Lag plot for a pseudorandom number generator with small m but relatively good properties for very short sequences.

3.2.3 *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 3.9 *Figure 3.9 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:

```
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 3.10. 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:*

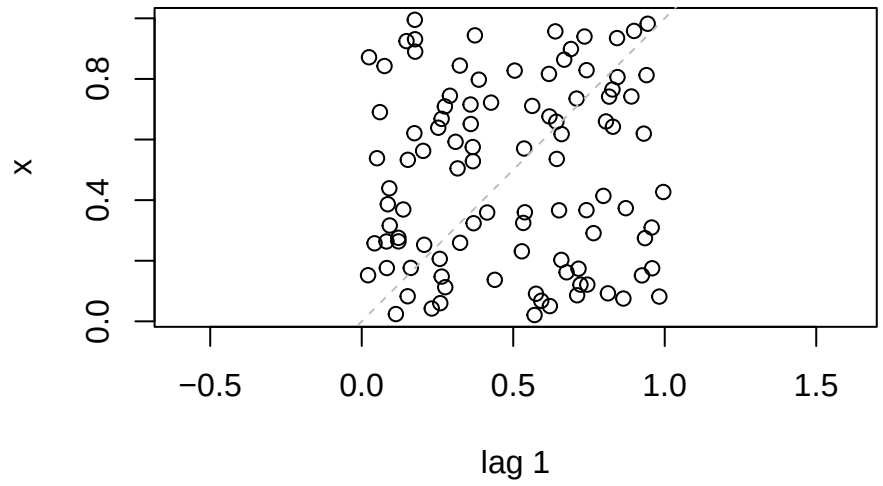


Figure 3.8: Lag plot for a pseudorandom number generator with moderate m and relatively good properties for short sequences.

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

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

Again, the autocorrelation function is plotted in Figure 3.11. 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 3.10 *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

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

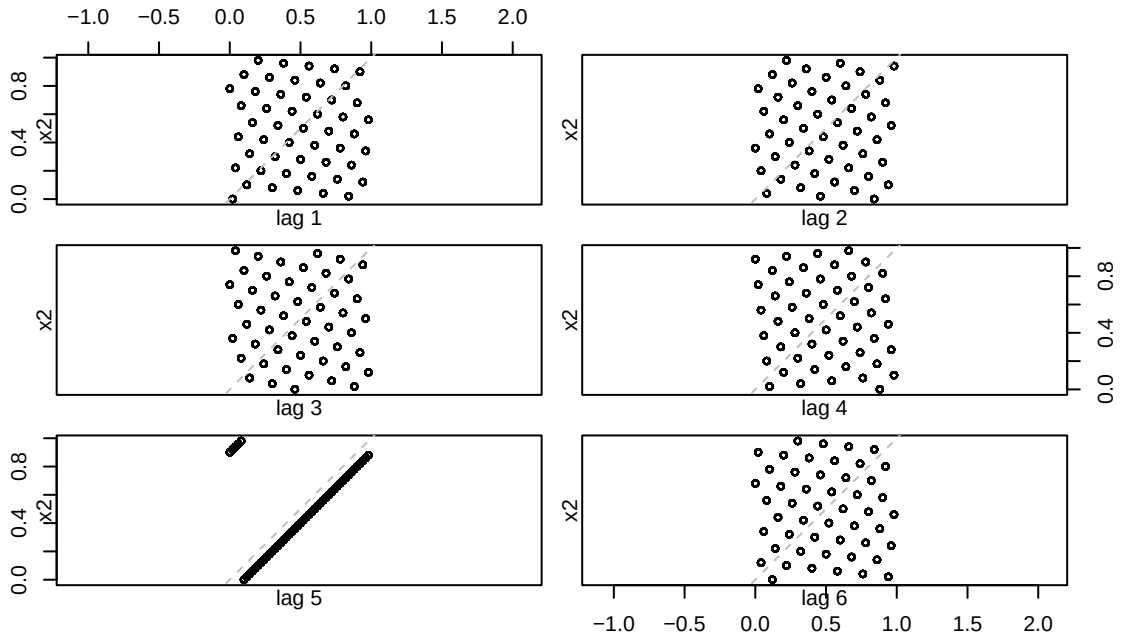


Figure 3.9: Lag plots for the first 6 lags.

```
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=""))
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 3.13. Here, we have used the `scatterplot3d` *function in the scatterplot3d package (Ligges and Mächler, (2003).*

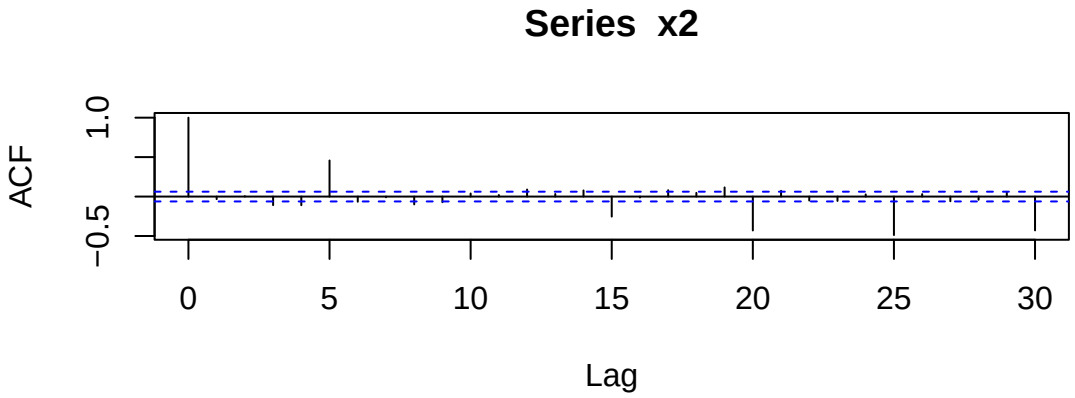


Figure 3.10: Autocorrelations for the x2 sequence.

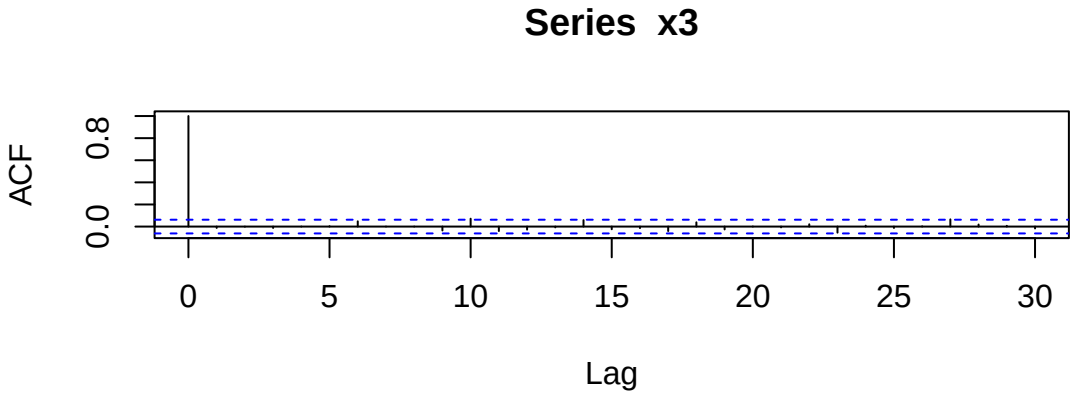


Figure 3.11: Autocorrelations for the x3 sequence.

```
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 3.13 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.

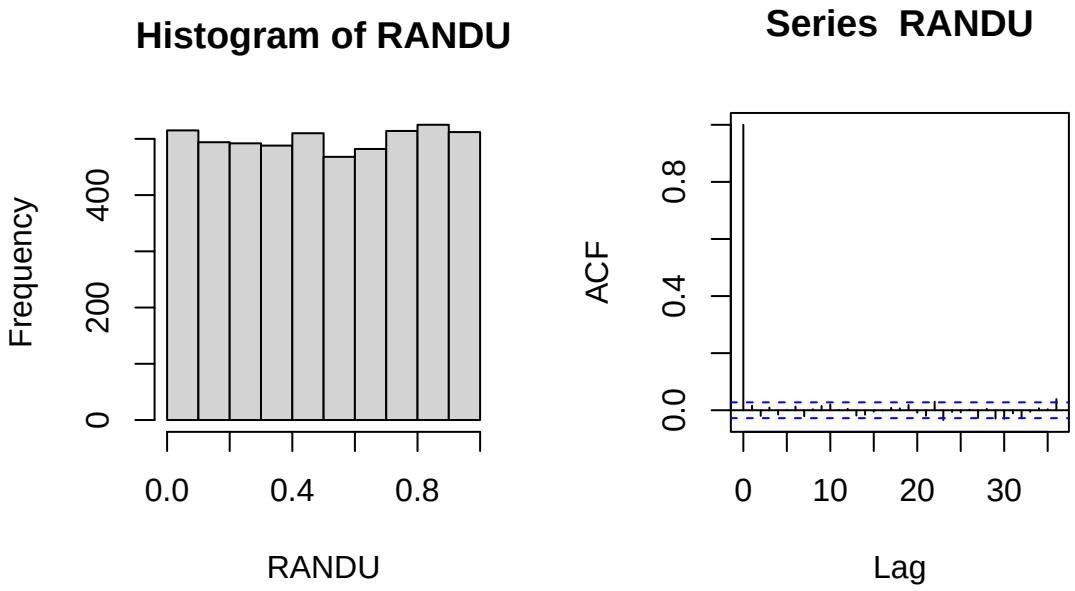


Figure 3.12: Histogram and ACF plots for a sequence of RANDU numbers.

3.3 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, such as in Chapter 2 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.

3.3.1 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 3.2.3, 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 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.

```
source("rftest.R")
```

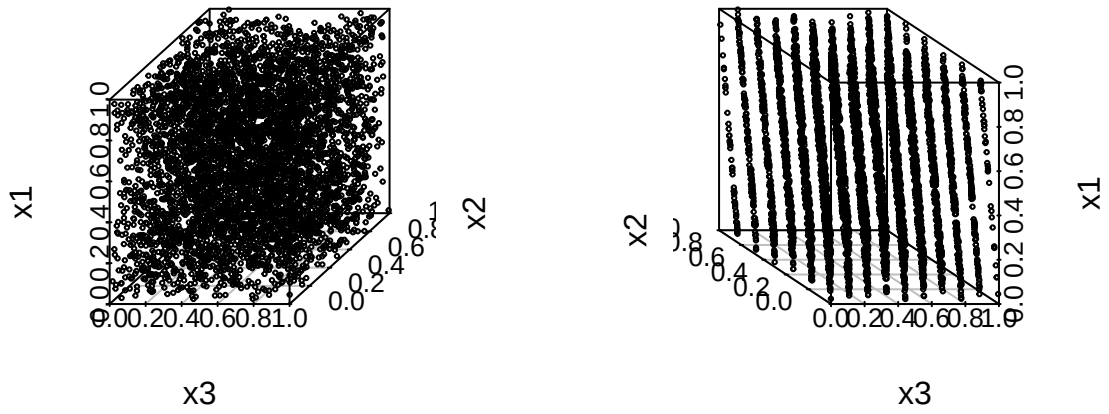


Figure 3.13: 3-dimensional lag plot of a sample from the RANDU simulator, from two different perspectives.

Example 3.11 *We start with the cosine sequence discussed in Section 3.1.2 to show why such a generator is not in practical use. The results of the random forest test are pictured in Figure 3.14 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 3.12 *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*.*

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

Execution of the following scripts yields the plots displayed in Figures 3.15 and 3.16. 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.*

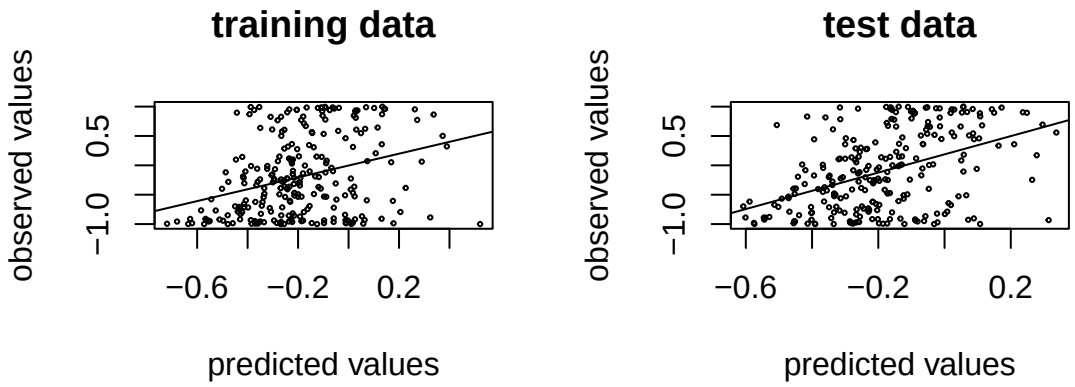


Figure 3.14: Random forest test for the cosine sequence of Section 3.1.2.

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

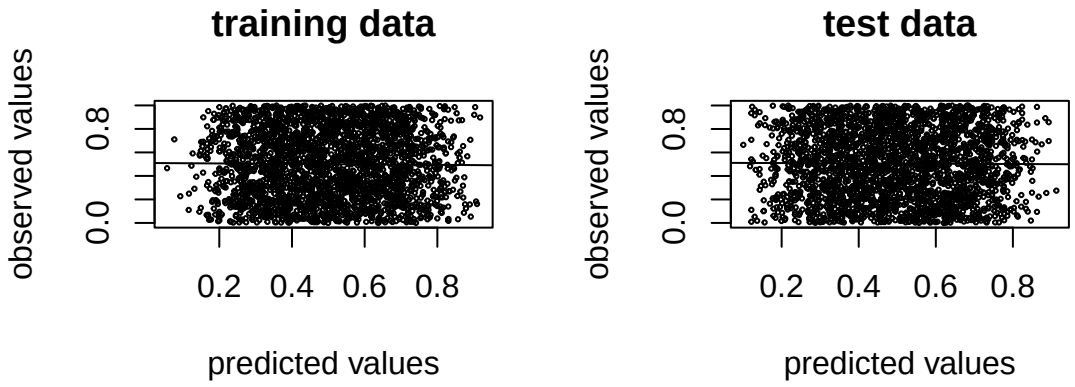


Figure 3.15: Random forest test for the default generator in R, using $m = 1$.

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

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

Example 3.13 *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 3.17 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

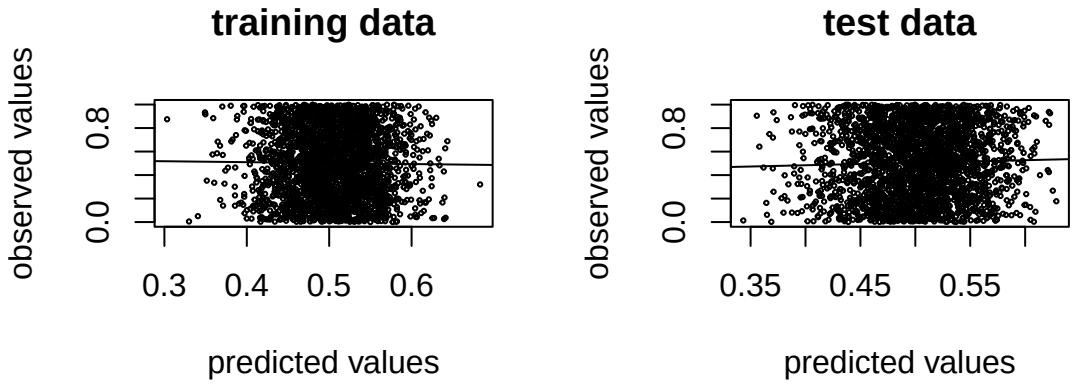


Figure 3.16: Random forest test for the default generator in R, using $m = 10$.

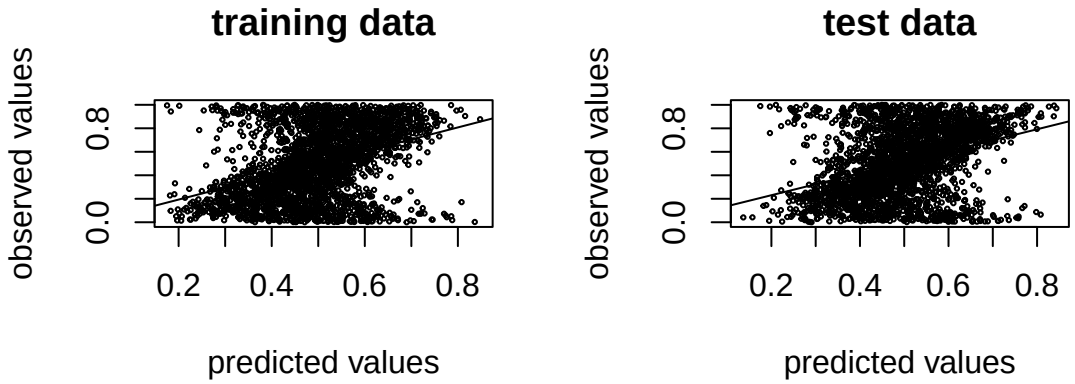


Figure 3.17: Random forest test for the RANDU sequence, using $m = 2$.

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.

3.4 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

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

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; $0 \leq x_0 < m$.

3.4.1 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.

Example 3.14 *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.

3.4.2 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
    }
    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 3.15 *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] 0.691 0.707 0.735 0.774

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 3.16 *Figure 3.18 displays a histogram and the autocorrelation function for the example of Section 3.4.2. 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)
```

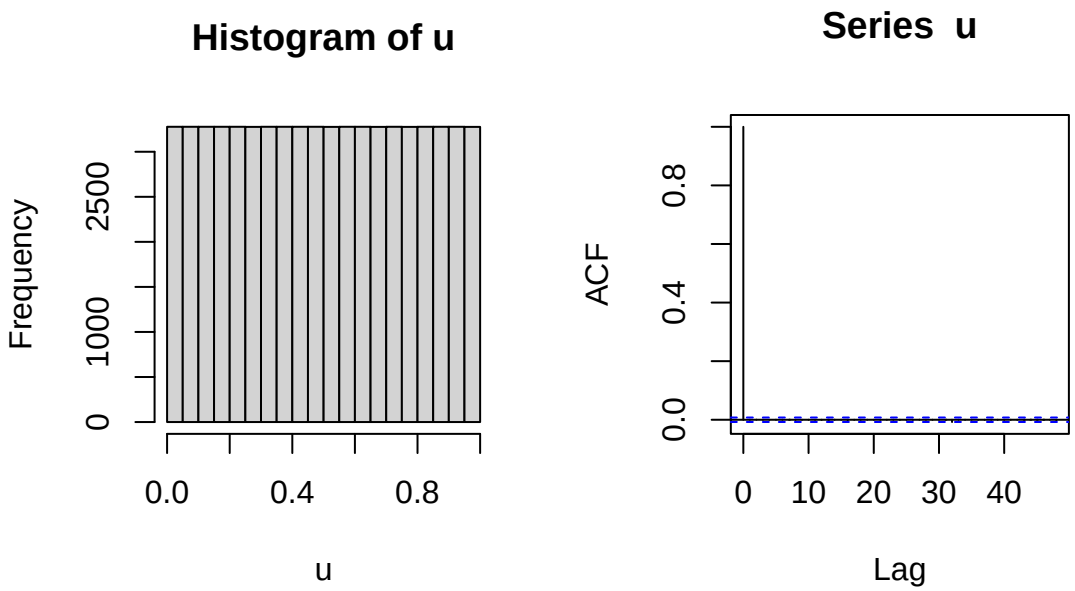


Figure 3.18: Histogram and ACF checks for the linear congruential sequence.

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:

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

Figure 3.19 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 3.20 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.

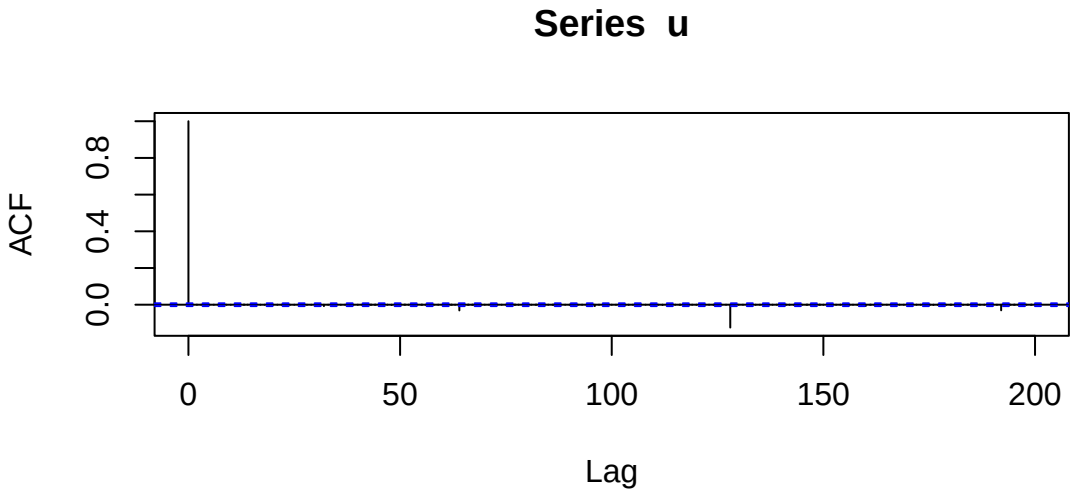


Figure 3.19: Histogram and ACF checks for the linear congruential sequence.

3.5 Other pseudorandom number generators

3.5.1 A quadratic congruential generator

The following scheme has been proposed as an alternative to linear congruential generators. Let $x_0 \bmod 4 = 2$, 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:

```
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 3.17 *We generate 1000 numbers using the default setting:*

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

The first few are

```
quadnumbers[1:8]
```

```
## [1] 0.873 0.644 0.209 0.688 0.655 0.183 0.848 0.723
```

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

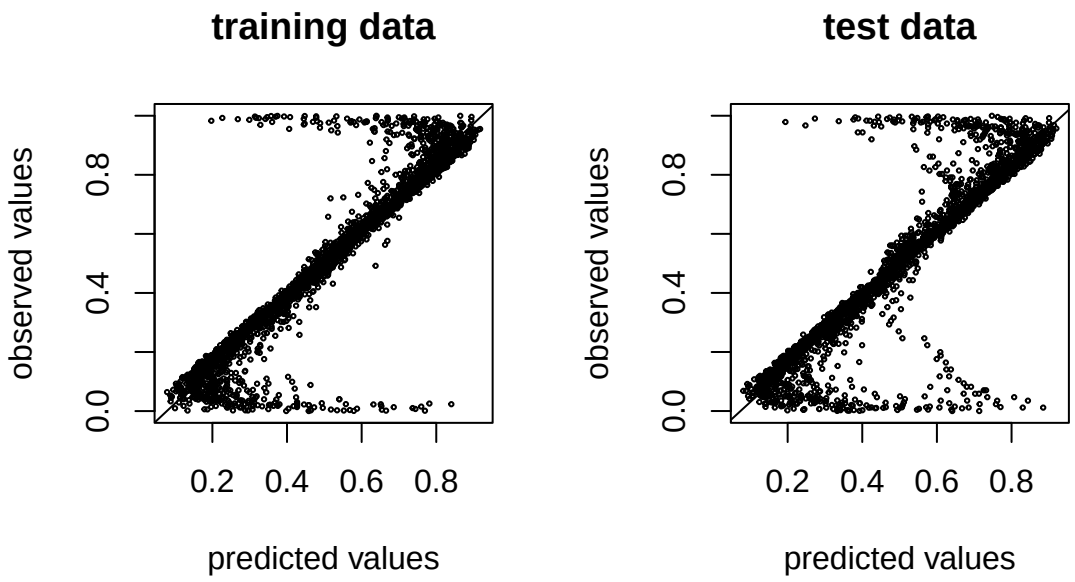


Figure 3.20: Random forest test for the linear congruential sequence, using $m = 5$.

Example 3.18 *Figure 3.21 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.*

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

Figure 3.22 reveals some autocorrelations that are very large.

3.5.2 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.

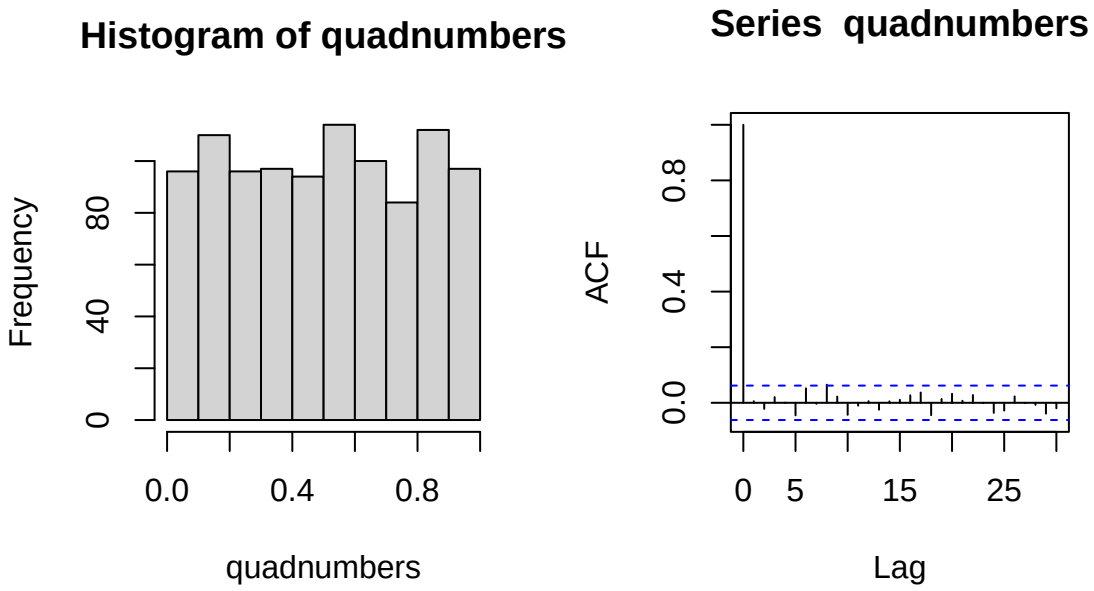


Figure 3.21: Histogram and ACF checks for the default quadratic congruential sequence.

3.5.3 *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.

3.5.4 *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 3.23 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 3.23 shows that the autocorrelations, even at very large lags, are all close to 0, a partial verification of this assertion.

```
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)
```

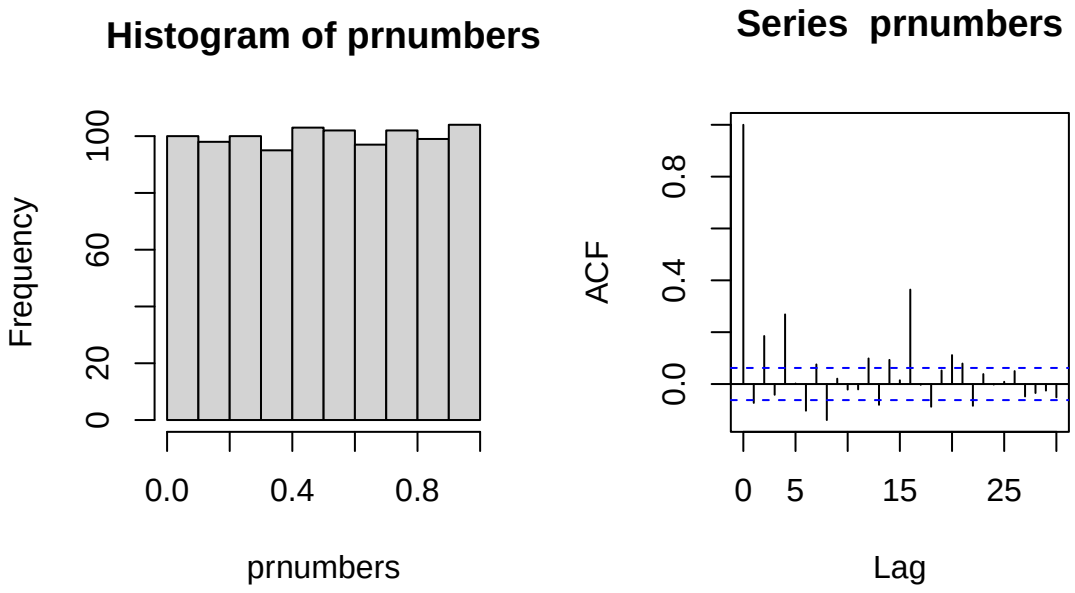


Figure 3.22: Histogram and ACF checks for another quadratic congruential sequence, with $\omega = 10$.

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.

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

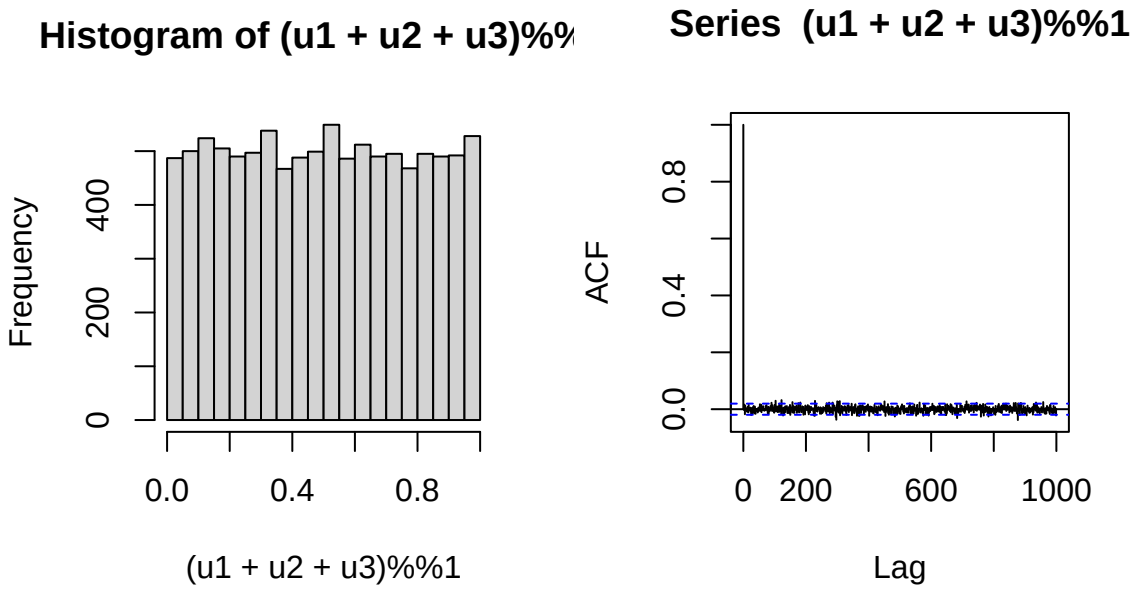


Figure 3.23: The fractional part of sums of independent uniforms is uniform.

Example 3.19 *We illustrate the use of the function to produce Wichman and Hill sequences here:*

```
rWH(4) # automatically generated seed

## [1] 0.347 0.267 0.921 0.477

rWH(4)

## [1] 0.0328 0.6016 0.7399 0.8619

rWH(4, 3329913) # our own seed

## [1] 0.637 0.401 0.493 0.261
```

Checking for autocorrelation and uniformity proceeds as before.

Example 3.20
```
par(mfrow=c(1,2))
prnumbers <- rWH(10000)
hist(prnumbers); acf(prnumbers)
```

Figure 3.24 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.

3.6 Default generator in R: the Mersenne twister

Maksumoto 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 $2^{19937} - 1$.

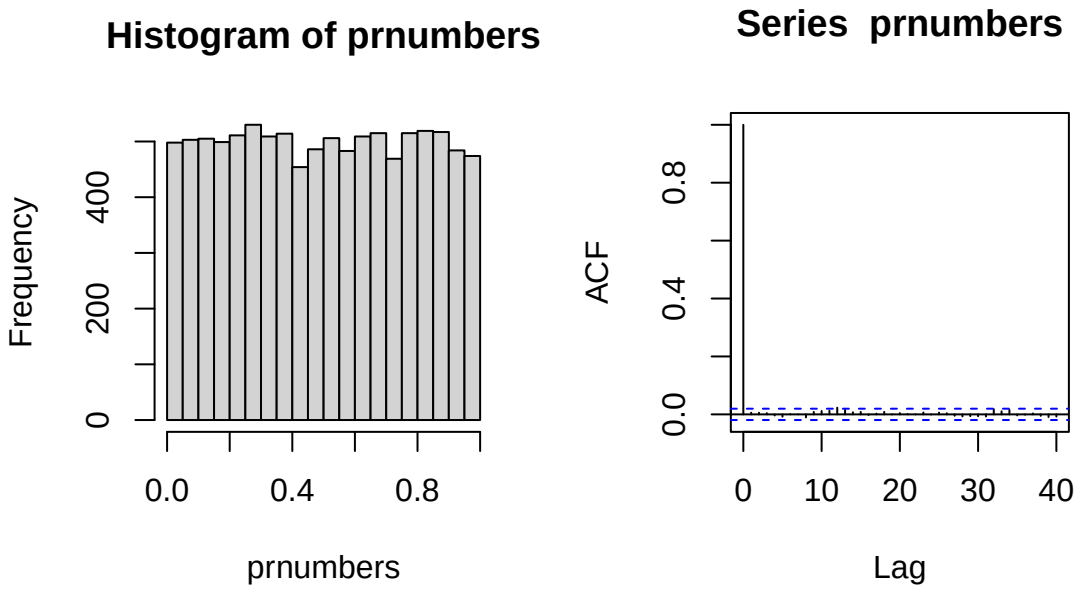


Figure 3.24: Histogram and ACF checks for the Wichman and Hill generator.

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

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

```
runif(3)

## [1] 0.102 0.254 0.539
```

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

##  [1]  0.306  1.807  2.147  0.918 -0.373  1.254  1.639 -0.924
##  [9]  2.470  0.334
```

3.7 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 3.22 *If you run the following code without specifying the seed, you will likely obtain values different from the 6 obtained below:*

```
runif(6)

## [1] 0.171 0.337 0.620 0.569 0.688 0.155
```

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] 0.652 0.733 0.836 0.290 0.358 0.963
```

3.8 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 3.23 ***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 3.25 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 3.26 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 3.24 *We can use the cross-correlation function to see how much dependence occurs between "hands". Figure 3.27 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)
}
```

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 $m \leq 10$ 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.

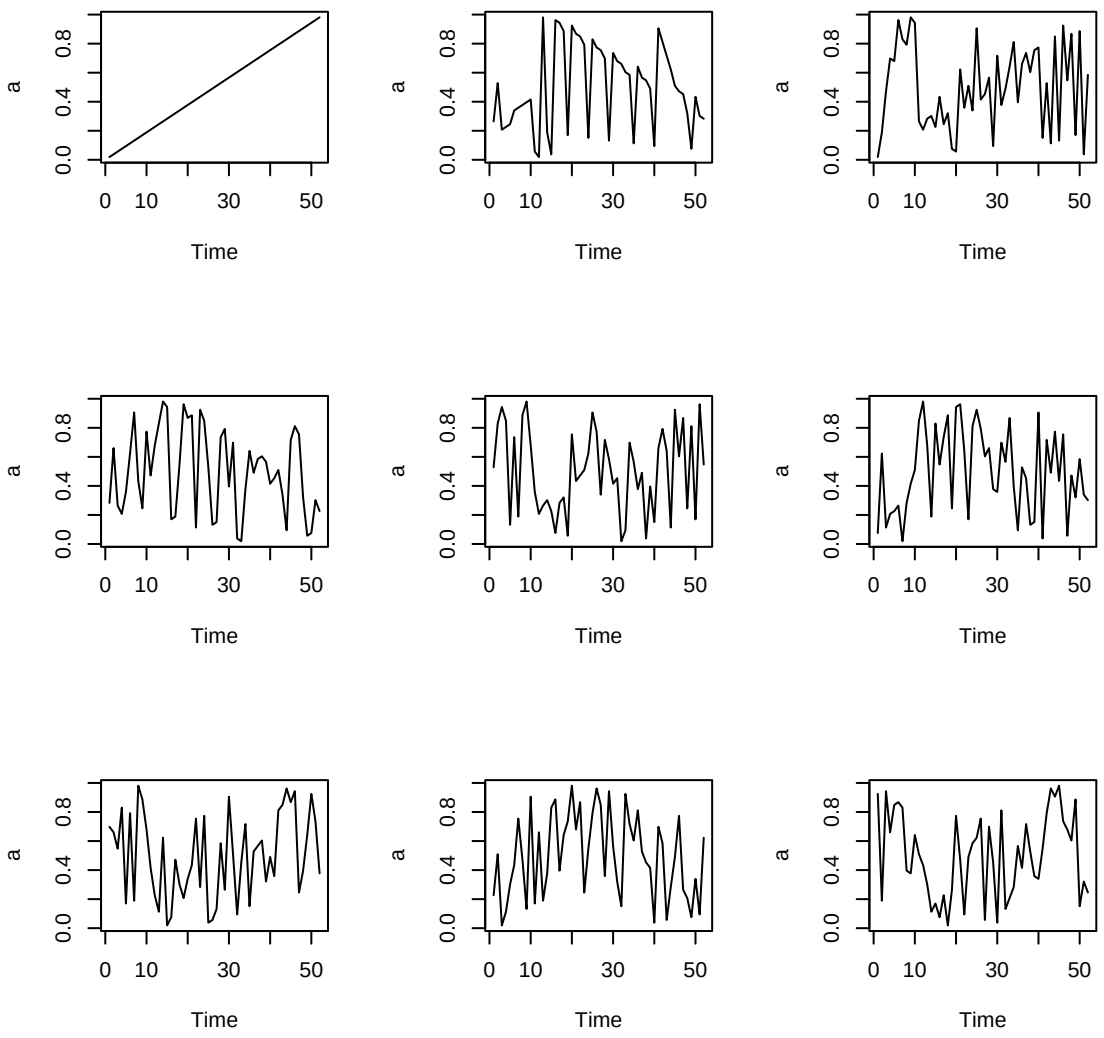


Figure 3.25: Trace plots for 9 successive shuffles of a deck of 52 numbers (representing playing cards in an ordinary deck).

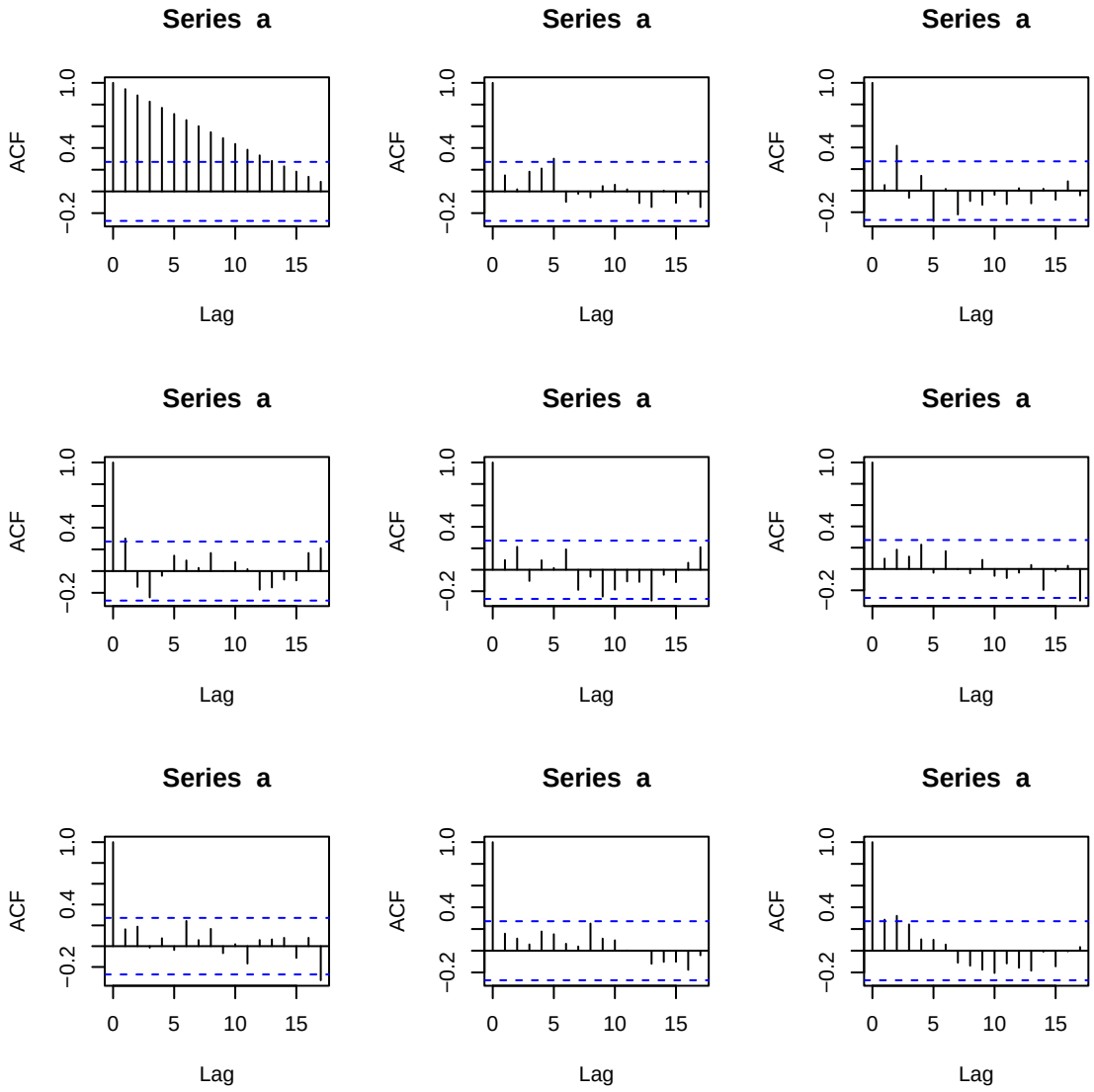


Figure 3.26: Autocorrelations for 9 successive shuffles of a deck of 52 numbers (representing playing cards in an ordinary deck).

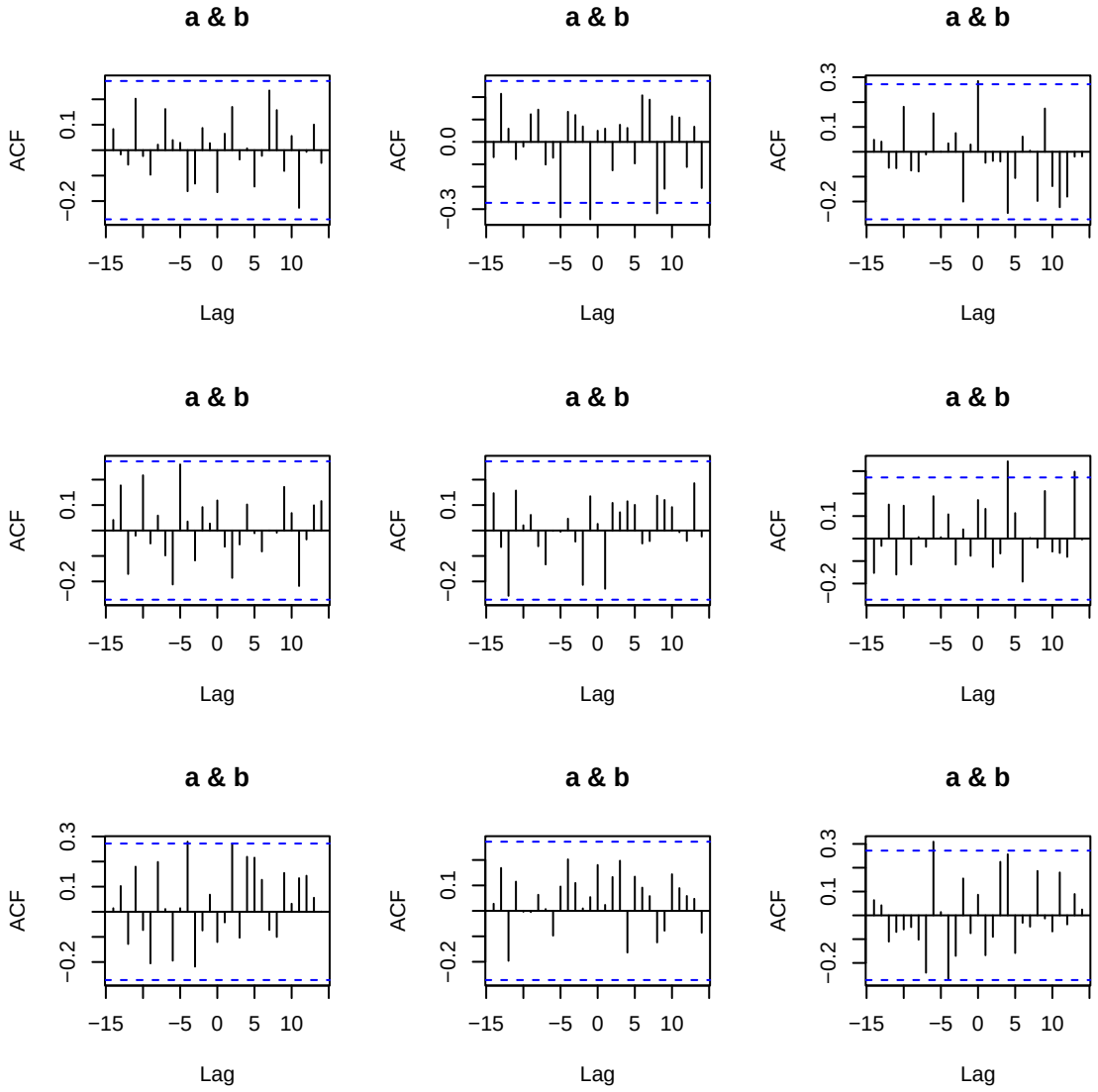


Figure 3.27: Cross-correlations between original ordering and first 9 orderings of a shuffled deck of 52 numbers (representing playing cards in an ordinary deck).

4

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.

4.1 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 4.1 *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 4.2 *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 4.3 *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 4.4 *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:

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

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

```
dFlaws(3)

## [1] 0.13
```

4.1.1 Visualizing a discrete distribution

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

Example 4.5 *Displaying the observed surface flaw distribution proceeds as follows:*

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

Figure 4.1 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.

4.1.2 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 4.6 *For the surface flaws example, we would estimate the cumulative distribution function as:*

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

For example, $P(F > 4)$

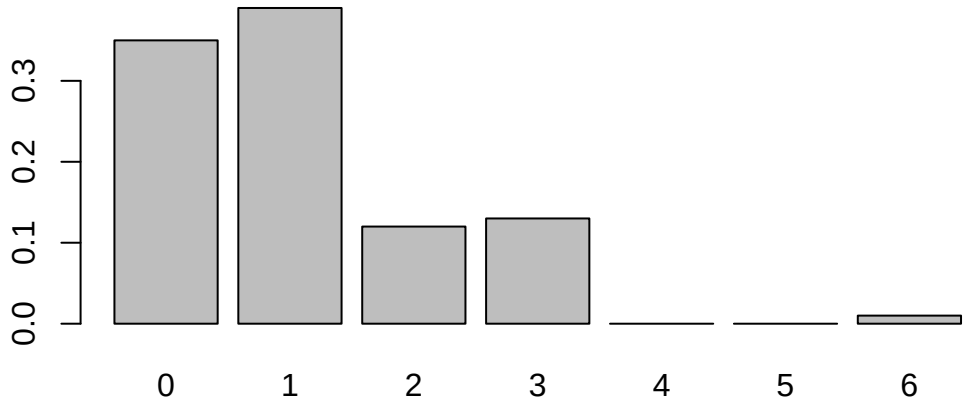


Figure 4.1: Bar plot of the observed surface flaw distribution.

```
1 - pFlaws(4)

## [1] 0.01
```

4.1.3 *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:

```
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 4.7 *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 3 2 3 1 0 0 1 1 3
```

Summarizing such data in a table is often appropriate

Example 4.8 *We simulate 100 values from the flaw distribution and tabulate the values as follows:*

```
table(rFlaws(100))

##
##  0  1  2  3
## 34 40 10 16
```

4.1.4 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] 3.850940e-01 3.265337e-01 1.948490e-01 7.594130e-02 1.629707e-02
## [6] 1.275522e-03 9.452461e-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       6
## 0.38291 0.32866 0.19413 0.07658 0.01641 0.00128 0.00003
```

Using the `sample` function

Example 4.9 *One application of* `sample()` *is to simple random sampling from a population:*

```
N <- 2000000 # population size
n <- 50 # sample size
sample(1:N, size = n, replace = FALSE)
```

We can also take a sample from a larger population:

```
##  [1] 1071132  411639  874144  460625  577906
##  [6] 1420385  493277 1864889 1830741 1042911
## [11] 1418223 1758078  701301 1316170  489990
## [16] 1990539 1707791 1829091  375078 1061350
## [21] 1291718 1941852  531192 1511320  489386
## [26]  712769  884640 1700323 1384806  305475
## [31]  480211 1440054  697549 1512200 1295398
## [36] 1982924  363324  201424 1560813 1689332
## [41] 1788033  838216  963324  677600  120968
## [46]  176811 1337610 1229163  963453  270377
```

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 4.10 *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 10 values can be obtained from the distribution using

```
sample(1:5, size = 10, replace = TRUE,
       prob = c(.1, .2, .1, .4, .2))

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

4.2 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 4.11 *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 4.12 *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] 1.015678
```

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

4.2.1 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 4.13 *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)
```

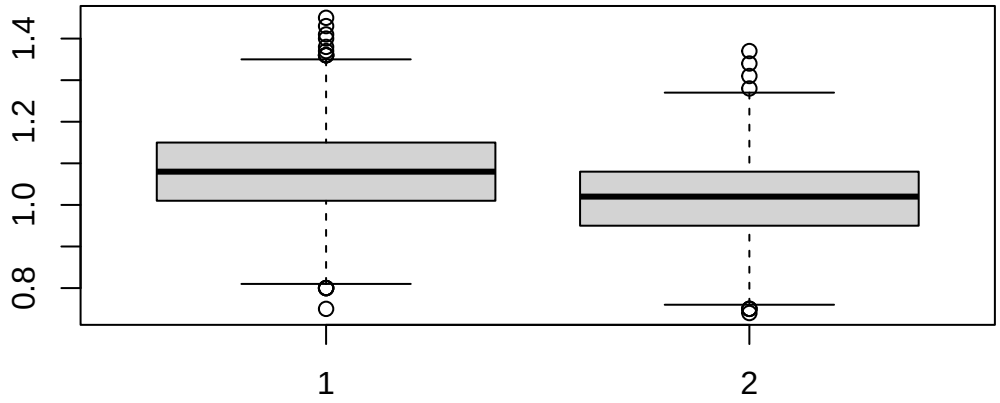


Figure 4.2: Side-by-side boxplots of the averages for the two surface flaw distribution models. The plot on the left is based on simulated samples coming from the data-based model. The plot on the right is based on the mathematical model.

Figure 4.2 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.

4.2.2 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 4.14 *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] 1.025076
```

4.2.3 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 4.15 *The standard error of the simulated values from the*

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

## [1] 0.1025076
```

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.

4.3 Special discrete random variable models

The following are the most commonly used discrete probability distributions.

- Bernoulli
- Binomial
- Geometric
- Poisson
- Negative Binomial

4.3.1 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.

4.3.2 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 4.16 *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] 0.7125406

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 $1 - p$.

4.3.3 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 $1 - p$:

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

4.3.4 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

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

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
##  [6] 0.57303 2.92813 1.33874 2.15108 0.83731
## [11] 0.43262 3.49134 3.57069 0.47193 1.50513
## [16] 1.24458 2.95007 0.85778 1.00540 2.12898
## [21] 1.26908 3.42819 2.77051 0.00868 3.48061
## [26] 2.52366 3.59945 3.19887 3.29401 1.09803
## [31] 1.70981 1.80608 1.21620 2.16889 0.15092
## [36] 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
##  [8] FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [15]  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
## [22] FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
## [29]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
## [36]  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 4.3 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")
```

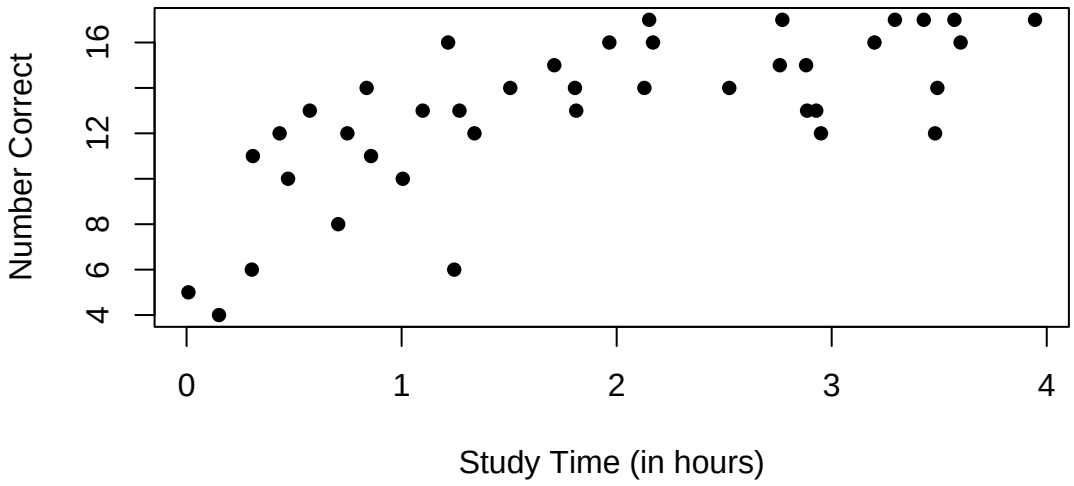


Figure 4.3: Scatter plot of simulated number of correct answers versus hours studied.

4.3.5 *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 4.17 *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 4.18

```
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))
```

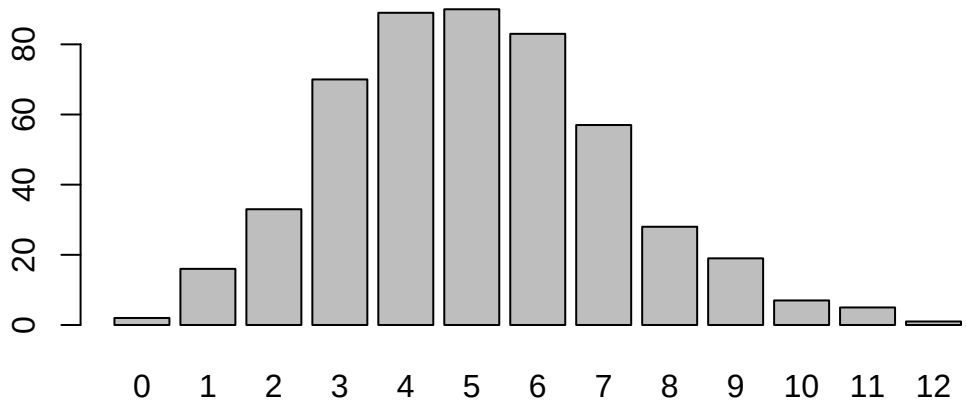


Figure 4.4: Bar plot exhibiting the distribution of a binomial random variable whose probability of success is 1/6.

Figure 4.4 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.

4.3.6 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 4.19 *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.

```
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
```

4.3.7 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 4.20 *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 4.21 *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
## [24] 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.

4.3.8 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 4.22 *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 4.23
```
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 4.24 *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 4.25 *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 4.5.

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

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.

4.3.9 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)
```

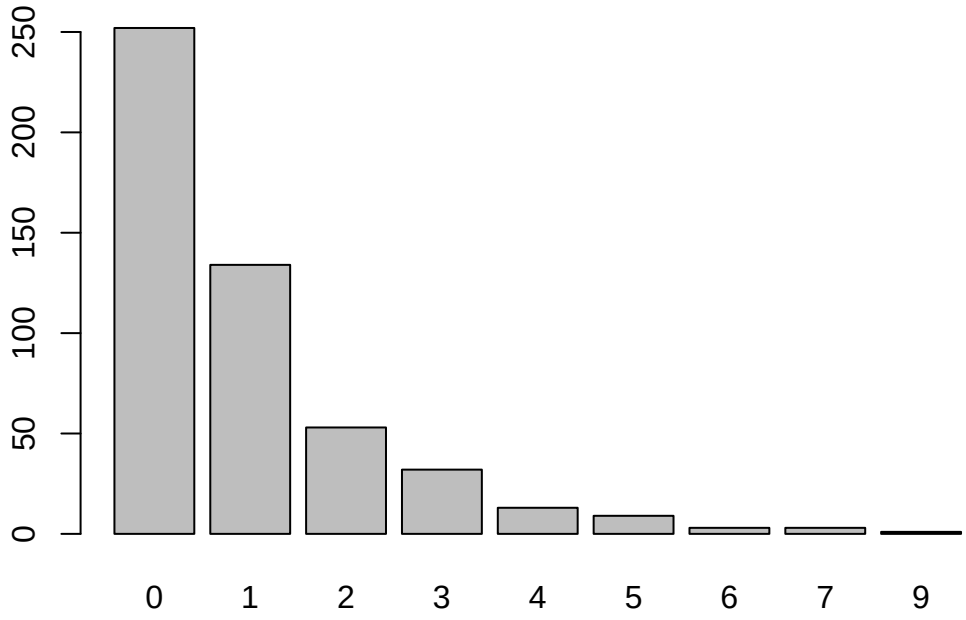


Figure 4.5: Bar plot of simulated geometric random variables.

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 4.26 *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 $p = \frac{1}{6}$.

4.3.10 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)
```

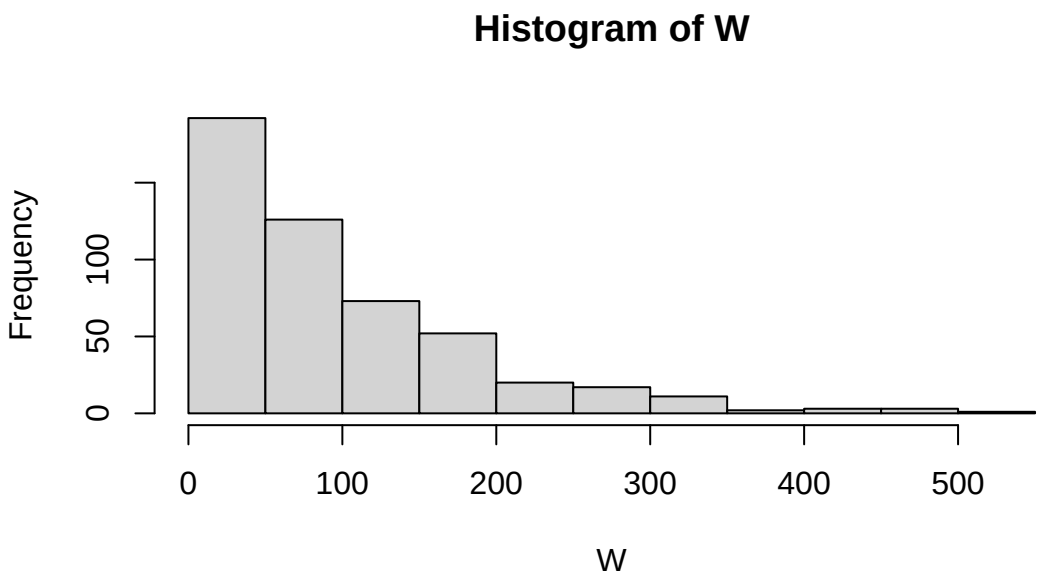


Figure 4.6: Histogram of simulated geometric random variables representing the waiting times until a 100-year event.

Figure 4.6 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 4.27 *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
```

4.3.11 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

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

4.4 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*.

4.4.1 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.

4.4.2 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 4.28 *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 4.29 `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
##  5007 14946 22474 22335 16755 10140  4962  2135
##     8     9    10    11    12    13
##   887   243    82    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
##  2.82932e-04  9.87949e-05  6.98192e-04
##             3             4             5
## -6.91808e-04 -4.81356e-04  5.81187e-04
##             6             7             8
## -7.89407e-04 -2.54031e-04  7.68488e-04
##             9            10            11
## -2.70504e-04  9.84882e-06  3.90497e-05
##            12            13
##  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.

4.4.3 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 4.30 *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
```

4.5 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 4.31 *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
##  [5] 2.3776094 2.7841103 3.7293372 5.0788597
##  [9] 6.0528535 6.7569535 6.8712531 7.8808272
```

4.6 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.

5

Simulation of Continuous Measurements

Figures 5.1 and 5.2 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.

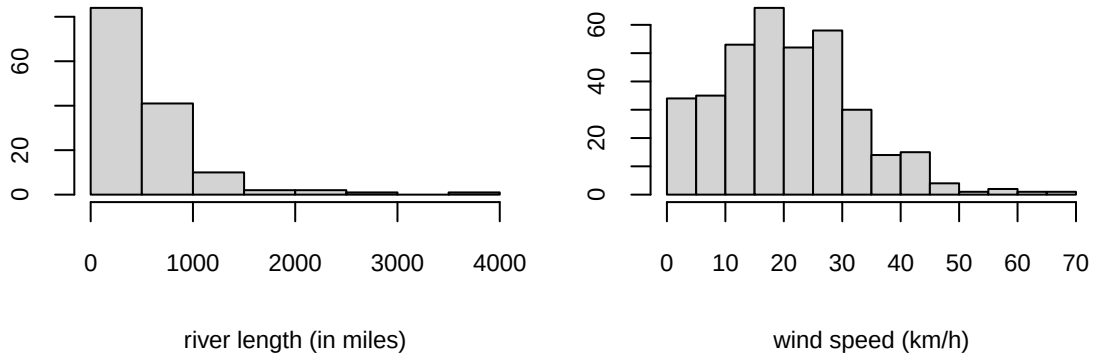


Figure 5.1: Left Panel: Histogram of lengths of long North American rivers. Right Panel: Histogram of noon hour wind speeds at Winnipeg International Airport

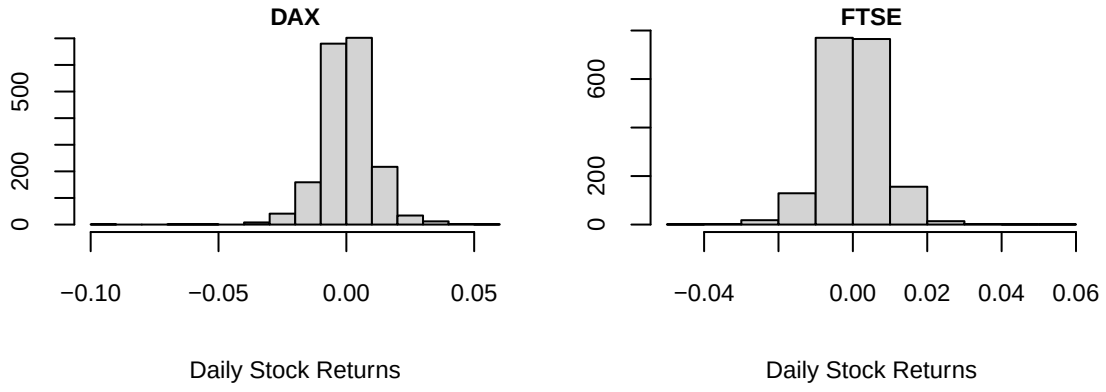


Figure 5.2: Histogram of stock returns for Germany's (DAX) and Britain's (FTSE) markets.

5.1 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.

5.1.1 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 5.1 *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 5.3. 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))
```

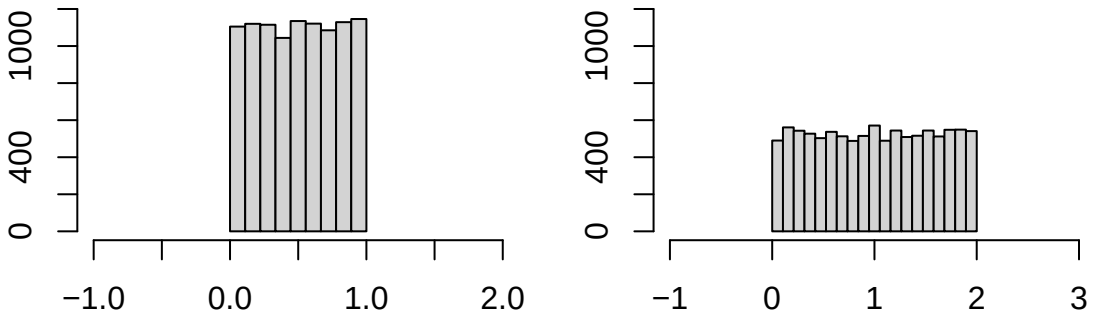


Figure 5.3: Histograms of 10000 simulated U and 2U values.

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 5.1 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 5.4 shows the relative frequency histograms for the simulated uniform samples from Example 5.1. 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)
```

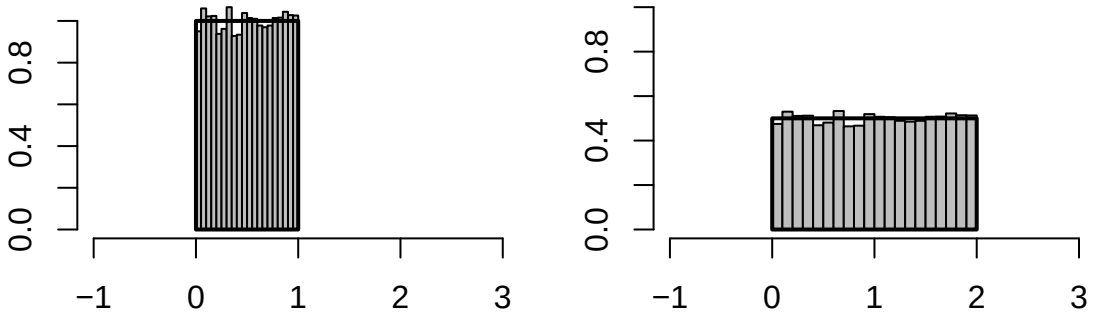


Figure 5.4: Relative frequency histograms of 10000 simulated U and 2U values with overlaid rectangles.

We have also overlaid each of the histograms in Figure 5.4 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 5.1.6.

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 5.2 *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 5.5. 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)
```

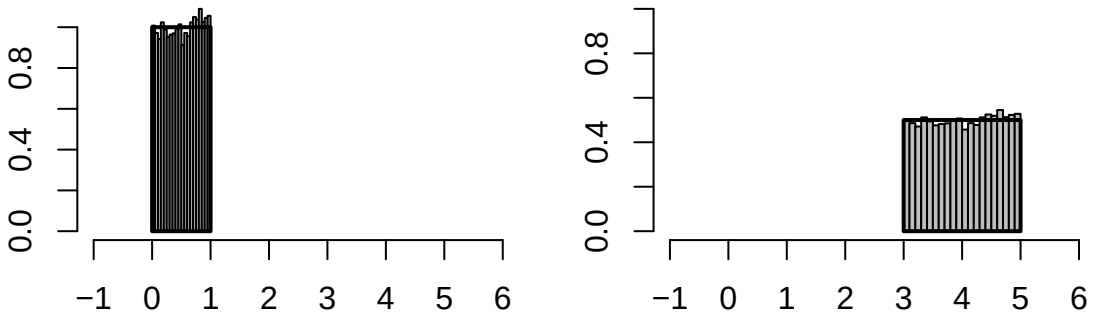


Figure 5.5: Relative frequency histograms of 10000 simulated U and $2U + 3$ values with overlaid rectangles.

This figure shows that after adding 3 to all values $2U$*, the resulting distribution is still uniform, but located on the interval* $[3, 5]$.

5.1.2 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 5.6 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 $x < 0$.

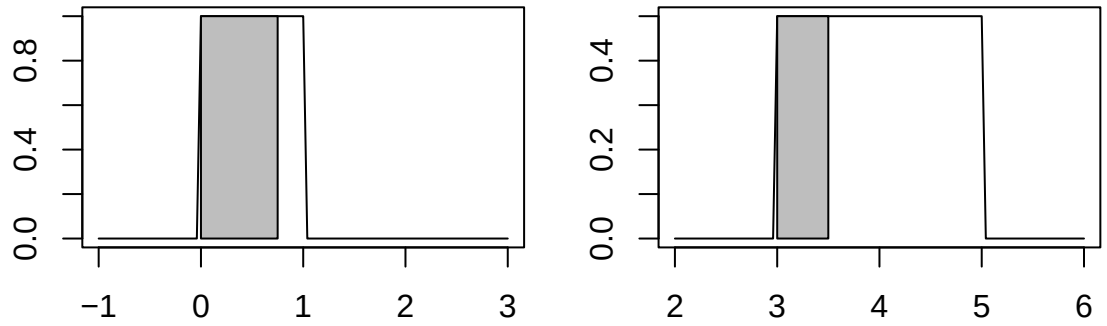


Figure 5.6: Shaded areas corresponding to $P(U \leq 0.75)$ and to $P(W \leq 3.5)$.

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 5.3 *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 5.4 *Returning to Example 5.3, 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 5.7.

```
curve(punif(x, 0, 1), -1, 6)
curve(punif(x, 3, 5), -1, 6)
```

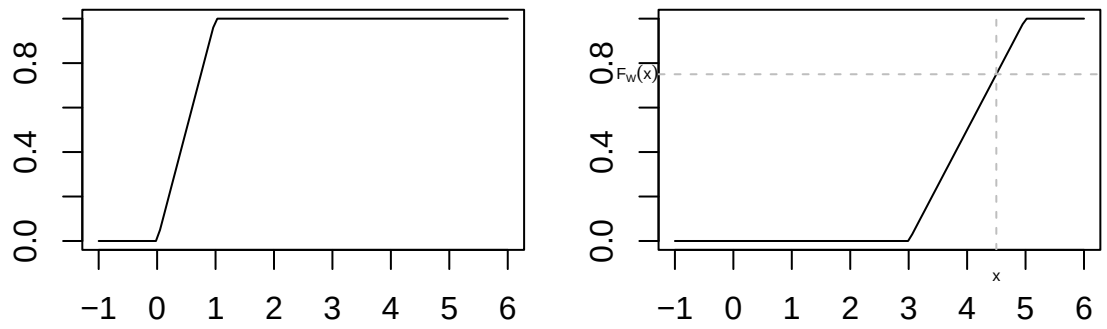


Figure 5.7: Graphs of the CDFs of U (left panel, $F_U(x)$) and $2U + 3$ (right panel, $F_W(x)$).

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 $F_W(4.5) = 0.75$.

5.1.3 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 5.5 *Figure 5.8 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:*

```
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.

5.1.4 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

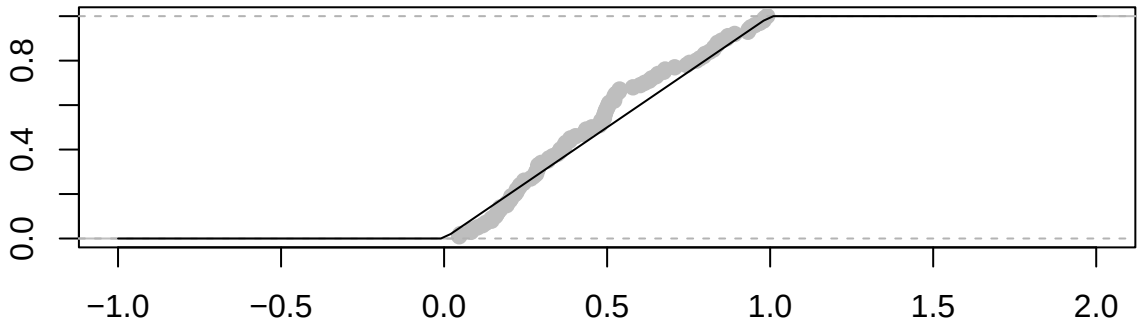


Figure 5.8: Histogram and cumulative histogram of 100 simulated U values. See Example 5.5.

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 5.6 *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 5.9. 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)
```

5.1.5 *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

$$F_U(q) = p. \tag{5.1}$$

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.

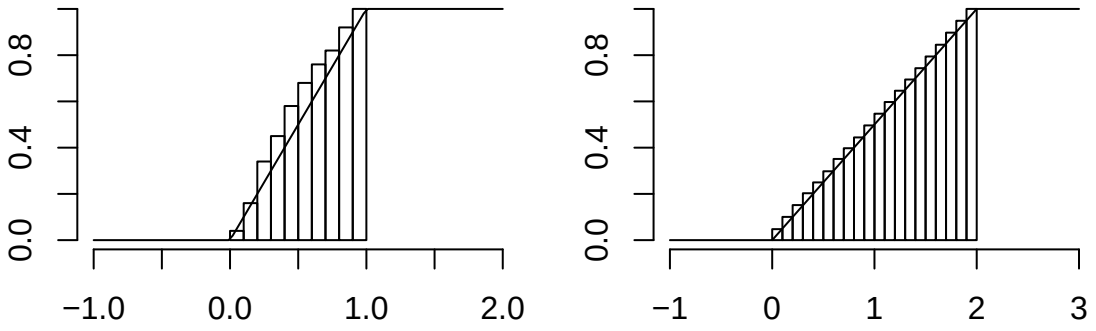


Figure 5.9: Cumulative histograms of 10000 simulated U and $V = 2U$ values with cumulative distribution functions overlaid.

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 (5.1) 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 $F_V^{-1}(p) = bp + a$.

5.1.6 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 5.6 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

$$P(W \leq x) = \frac{x - a}{b}, \quad \text{for } x \in [a, a + b].$$

5.1.7 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 5.7 *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 5.8 *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.

5.1.8 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 5.9 *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 5.10 *For the uniform sample simulated in Example 5.8, 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*.*

5.2 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.

5.2.1 *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 5.11 *If* $V = U^2$*, we obtain a relative frequency histogram as in Figure 5.10. 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="")
```

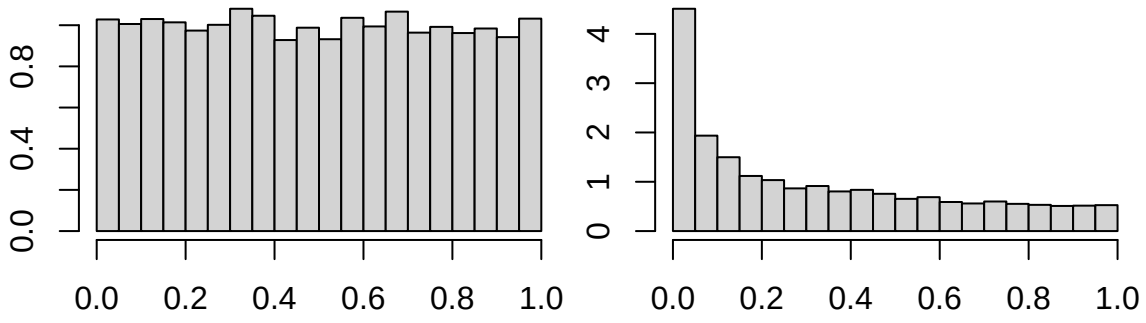


Figure 5.10: Relative frequency histograms of 10000 simulated U (left panel) and corresponding U^2 (right panel) values.

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 5.12 *Figure 5.11 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)
```

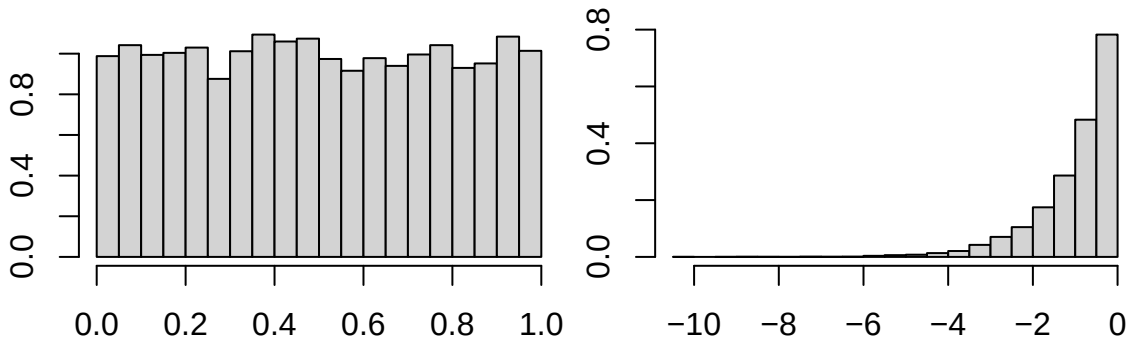


Figure 5.11: Histograms of 10000 simulated U and corresponding $\log(U)$ values.

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 5.12, 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)
```

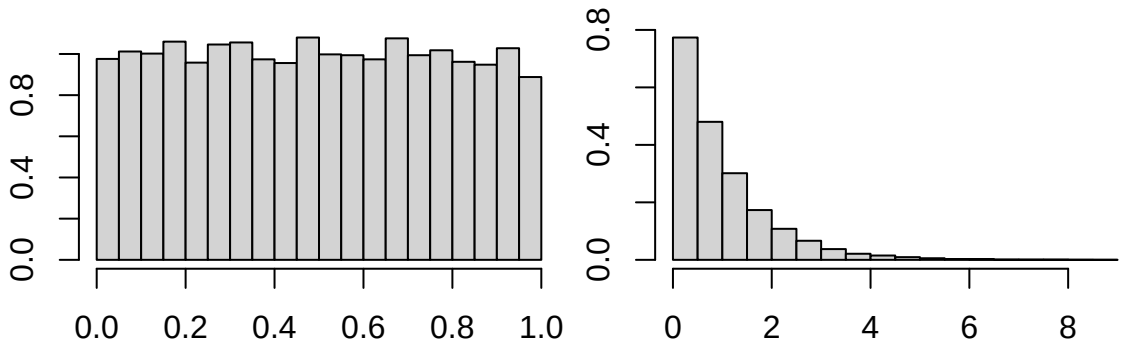


Figure 5.12: Histograms of 10000 simulated U and corresponding $-\log(U)$ values.

5.2.2 *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

$$F_U(x) = P(U \leq x) = \begin{cases} x, & 0 < x \leq 1 \\ 1, & x > 1 \\ 0, & x \leq 0 \end{cases} \tag{5.2}$$

We have used the distribution function defined at (5.2) 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 5.13 *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 (5.2) 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

$$F_V(x) = P(V \leq x) = \begin{cases} \sqrt{x}, & 0 < x \leq 1 \\ 1, & x > 1 \\ 0, & x \leq 0 \end{cases} \tag{5.3}$$

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 5.14 *Returning to Example 5.13, 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 5.15 *Returning to Examples 5.11 and 5.13, 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 5.13. The figure in the right panel is of the relative cumulative histogram of the simulated data, with the CDF overlaid.*

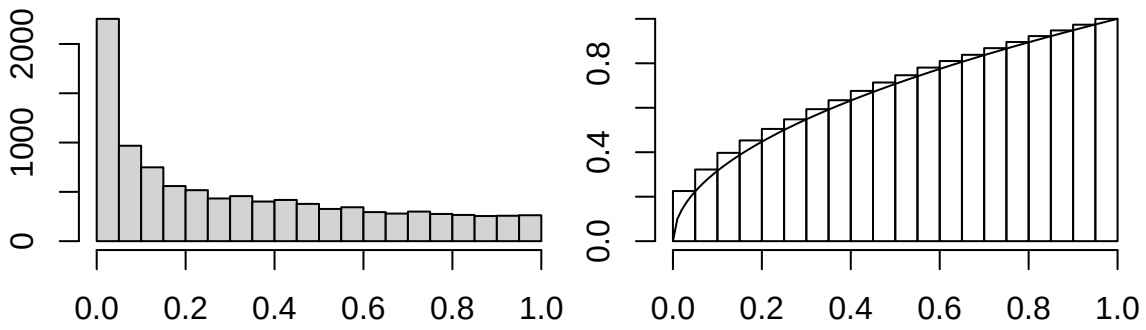


Figure 5.13: Histogram and cumulative histogram of 10000 simulated V values. See Example 5.11.

The right panel of the figure shows that the sample cumulative histogram and the true CDF match very well.

Example 5.16 *If $W = \log(U)$, we may obtain the distribution function by starting from:*

$$F_W(x) = P(W \leq x) = P(\log(U) \leq x).$$

This time, we note that $\log(U)$ will be less than or equal to x when, and only when, $U \leq e^x$. Therefore,

$$F_W(x) = P(U \leq e^x) = e^x$$

when $x < 0$, and $F_W(x) = 1$, when $x \geq 0$.

Figure 5.14 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)
```

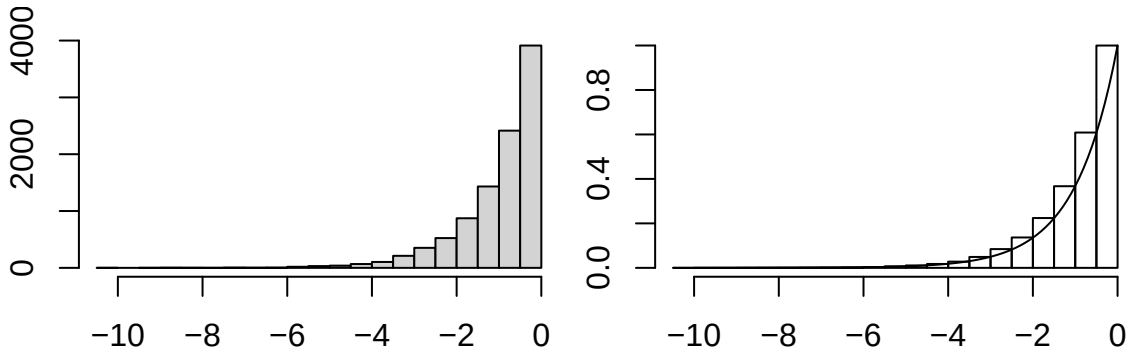


Figure 5.14: Histogram and cumulative histogram of 10000 simulated W values. See Example 5.12.

5.2.3 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

$$F_V(x) = P(V \leq x) = P(g(U) \leq x) = P(U \leq g^{-1}(x)) = g^{-1}(x),$$

for $x \in [0, 1]$. If $g(x)$ is a decreasing function, such as $-\log(x)$, then the inequality above is flipped:

$$F_V(x) = P(V \leq x) = P(g(U) \leq x) = P(U \geq g^{-1}(x)) = 1 - g^{-1}(x),$$

for $x \in [0, 1]$.

5.2.4 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 5.17 *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 5.15.

```
hist(V); chist(V); curve(sin(x), 0, pi/2, add=TRUE)
```

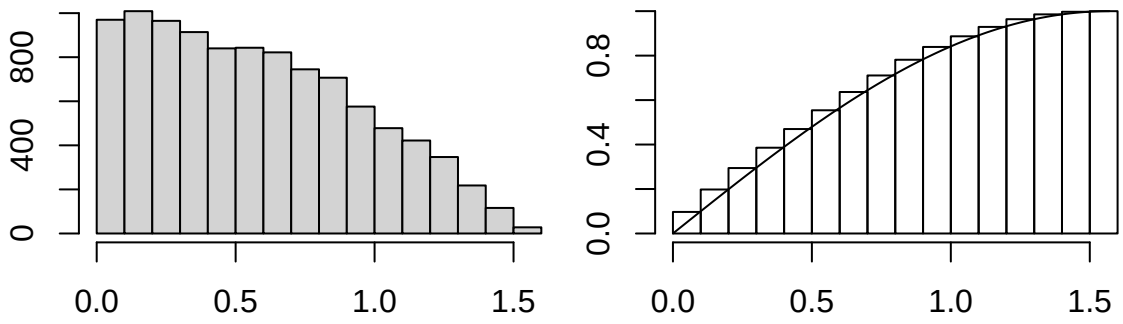


Figure 5.15: Histogram and cumulative histogram of 10000 simulated V values. See Example 5.17.

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 5.18 *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 5.16. 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)
```

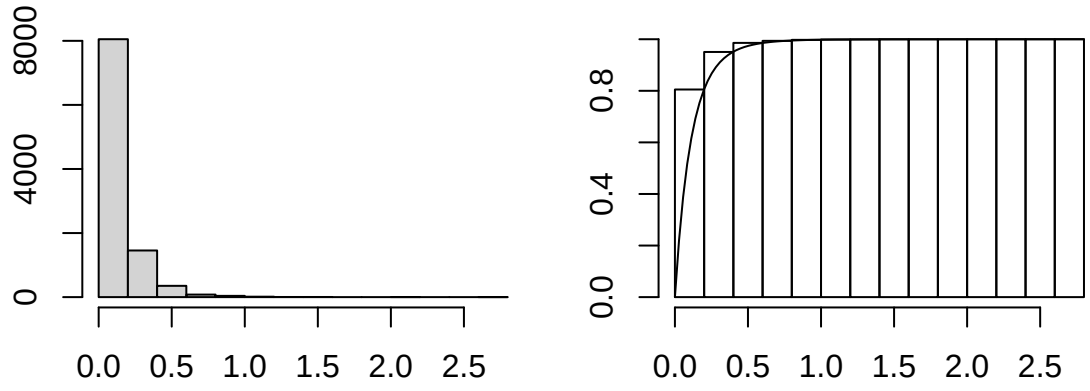


Figure 5.16: Histogram and cumulative histogram of 10000 simulated V values. See Example 5.18

Again, the sample cumulative histogram and the true CDF match very well. Notice that the sample has a right-skewed distribution.

Example 5.19 *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 5.12. 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 $x \in [0,1]$,*

$$P(1-U \leq x) = P(U \geq 1-x) = 1-(1-x) = x = P(U \leq x).$$

5.2.5 *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 5.20 *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 5.21 *The 15th and 75th percentiles of the distribution of V from Example 5.18 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 5.22 *A QQ-plot for the data simulated in Example 5.18 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 5.17 with a reference line, having 0 intercept and slope 1, overlaid.

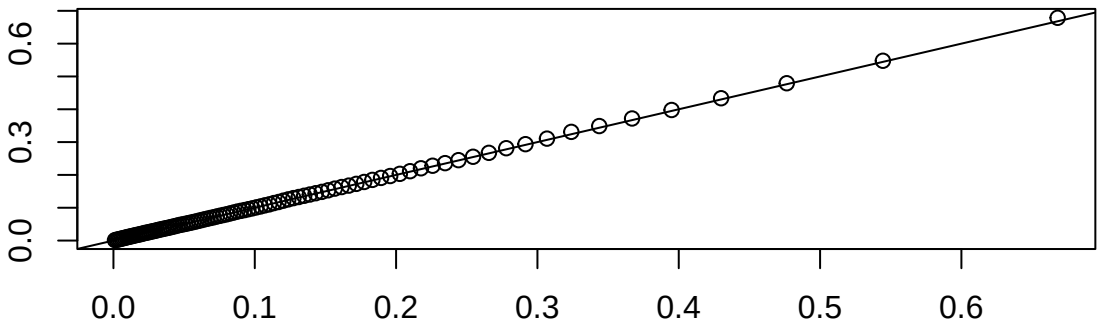


Figure 5.17: QQ-plot.

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.

5.2.6 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

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx.$$

Such probabilities are also expressed in terms of the *cumulative distribution function*:

$$F_X(y) = P(X \leq y) = \int_{-\infty}^{y} f_X(x)dx.$$

Example 5.23 *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 $v \geq 1$, 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 5.24 *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 5.25 *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 5.18. 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)
```

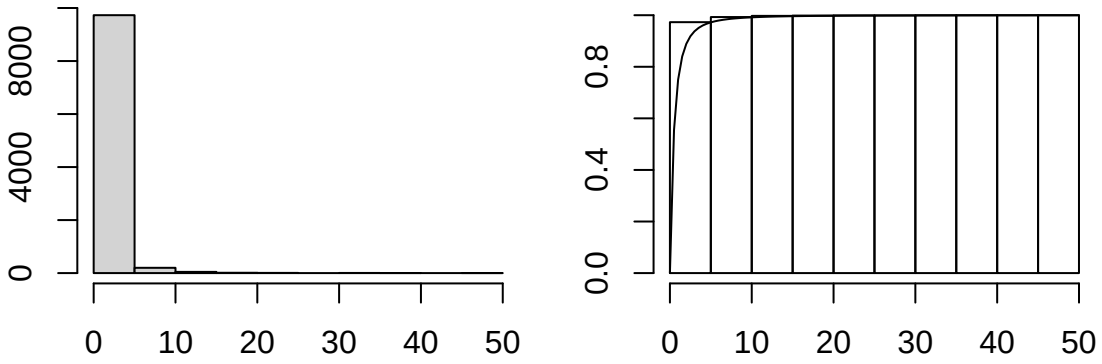


Figure 5.18: Histogram and cumulative histogram of 10000 simulated V values. See Example 5.25.

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.

5.2.7 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 5.26 *For the probability density function $f_X(x) = (\alpha+1)x^\alpha$, for $x \in [0,1]$, we have*

$$E[X] = \int_0^1 x(\alpha+1)x^\alpha dx = \frac{\alpha+1}{\alpha+2}. \tag{5.4}$$

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 5.27 *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 5.28 *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 5.29 *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 5.30 *Consider the distribution in Example 5.29, 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 5.28.

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 5.31 *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 5.32 *Consider the PDF of Example 5.26 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 5.33 *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.

5.2.8 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 5.34 *For the PDF of Example 5.26, $f_X(x) = (\alpha+1)x^\alpha$, 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 $\text{Var}(X)$ implies that there is more certainty about the value of X; it will tend to take values close to μ when $\text{Var}(X)$ is very small. The distribution will be more spread out when $\text{Var}(X)$ 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

$$\text{Var}(aX) = a^2\text{Var}(X) \tag{5.5}$$

for any nonrandom constant a, and

$$\text{Var}(X+a) = \text{Var}(X). \tag{5.6}$$

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 5.35 *From Examples 5.27 and 5.33, 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.*

5.2.9 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 5.36 *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 5.37 *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 5.38 *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 5.17. 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.

5.3 Simulating from the family of normal models

5.3.1 Why does the normal distribution occur?

We encountered simulated normal random variates back in Example 1.2, arising as a sum of independent uniform random numbers. Normality arises from summing many other kinds of random variables.

Example 5.39 *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 5.19.

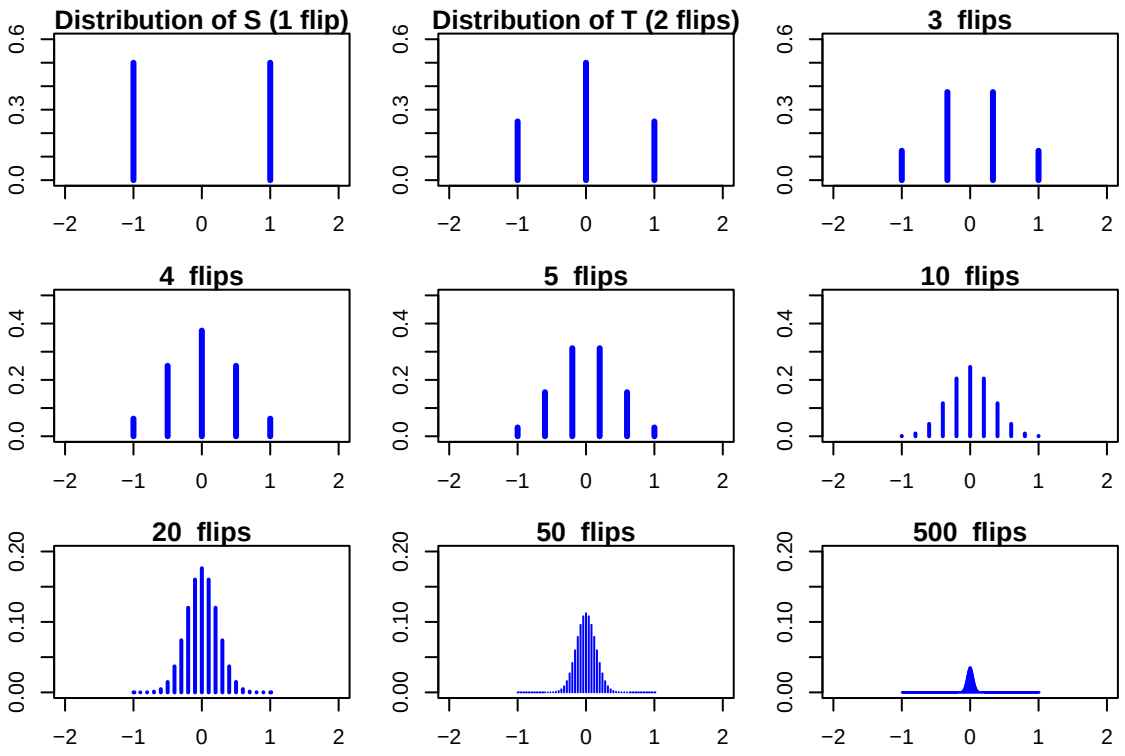


Figure 5.19: Distributions of coin flip game averages for various numbers of flips. Each flip counts as either $+1$ or -1.

The other panels of Figure 5.19 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 5.40 *We now simulate data coming from the distribution pictured in the top left panel of Figure 5.20. 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.

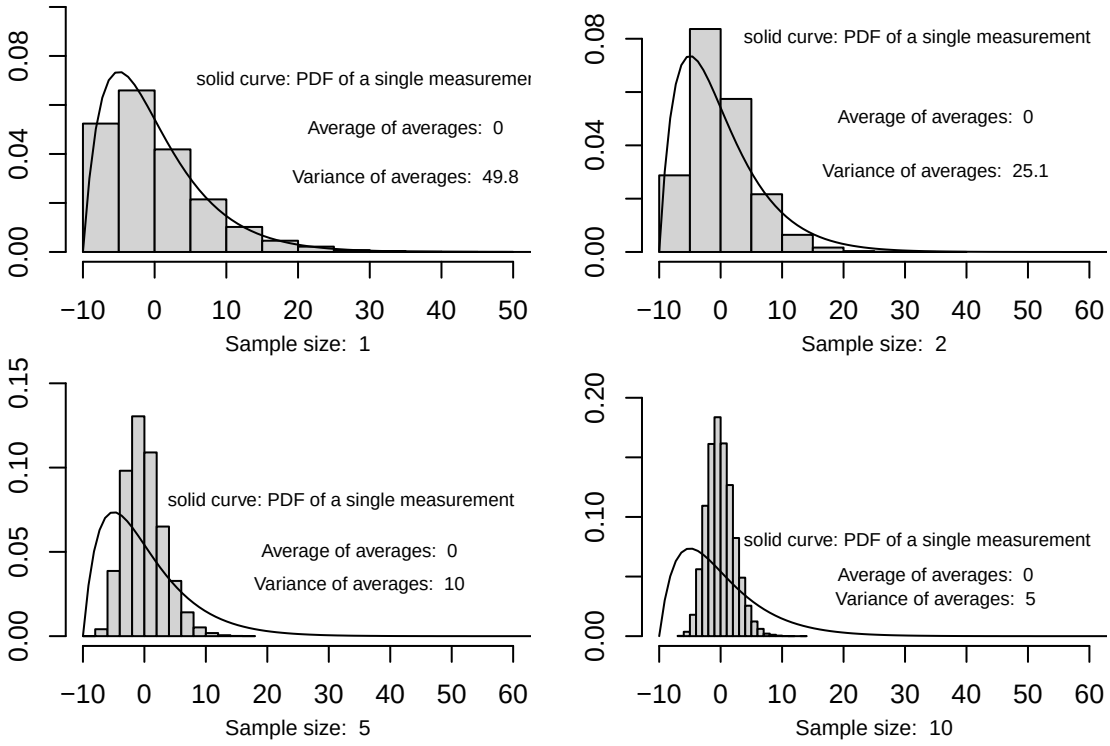


Figure 5.20: Histograms of averages of simulated samples of different sizes from a skewed distribution. The PDF for the distribution is overlaid on each histogram to emphasize that the data are originally coming from such a population.

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 1.2 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 5.41 *As in Example 5.39, 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 5.42 *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.

5.3.2 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()`.

5.3.3 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)
```

5.3.4 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 5.21 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.

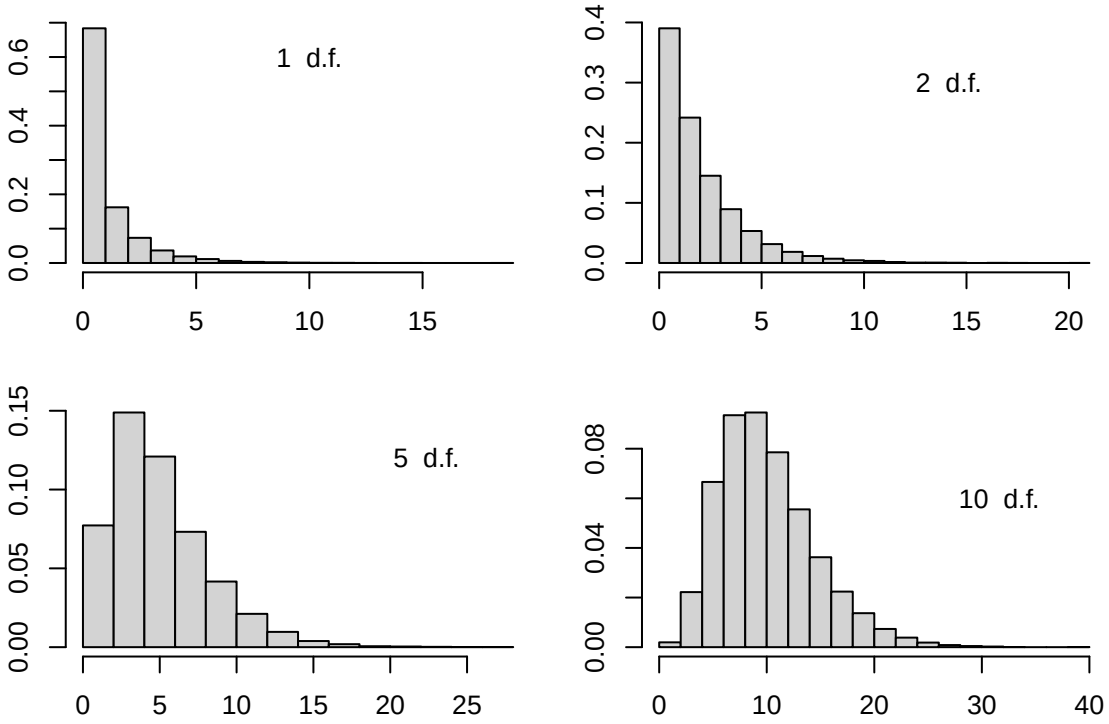


Figure 5.21: Histograms of simulated chi-squared random numbers on 1, 2, 5, and 10 degrees of freedom.

5.4 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 5.22.

```
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,

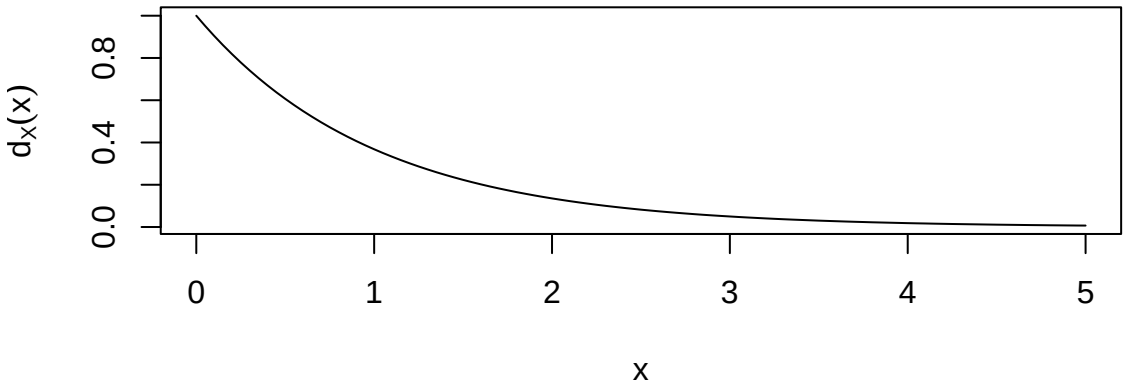


Figure 5.22: PDF of the exponential distribution with rate 1.

```
X <- rexp(9); X

## [1] 2.584 1.308 1.329 1.232 0.597 1.266 0.459 1.228 8.319
```

5.4.1 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 5.23 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
```

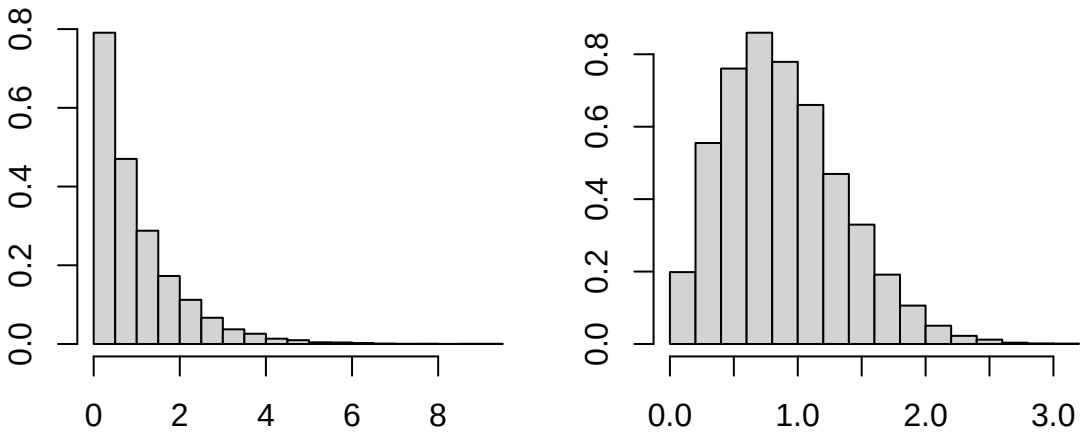


Figure 5.23: Histograms of exponential random variates (left panel) and their square roots (right panel) - Weibull variates with shape parameter 2.

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 5.24 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)
```

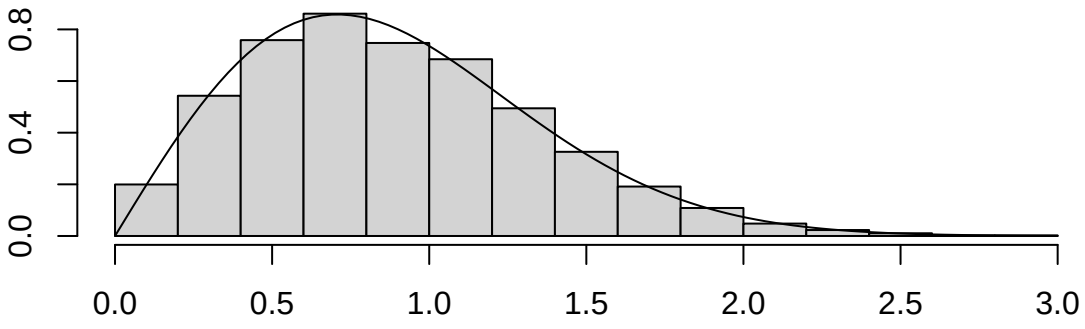


Figure 5.24: Histogram of simulated Weibull data with overlaid Weibull PDF.

5.4.2 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

5.25.

```
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)
```

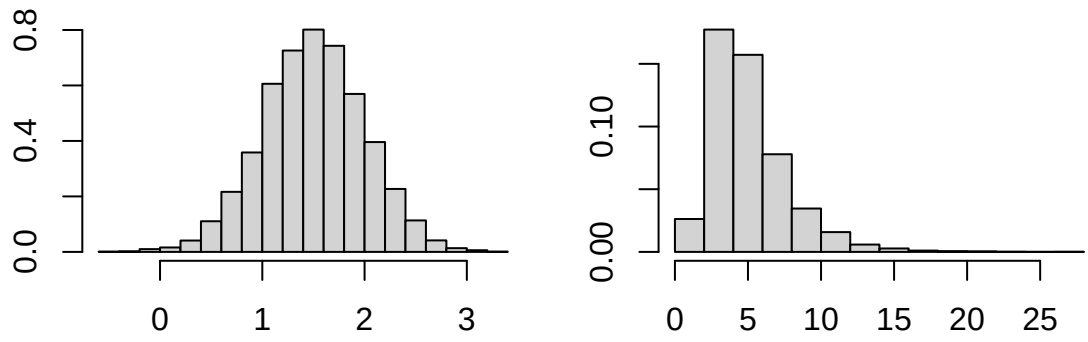


Figure 5.25: Histograms of simulated normal and corresponding lognormal data (right panel).

We now illustrate the effectiveness of a simple model like the lognormal distribution on some medical data.

Example 5.43 *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 5.26. The raw data are in the left panel. The log of the raw data are in the right panel.

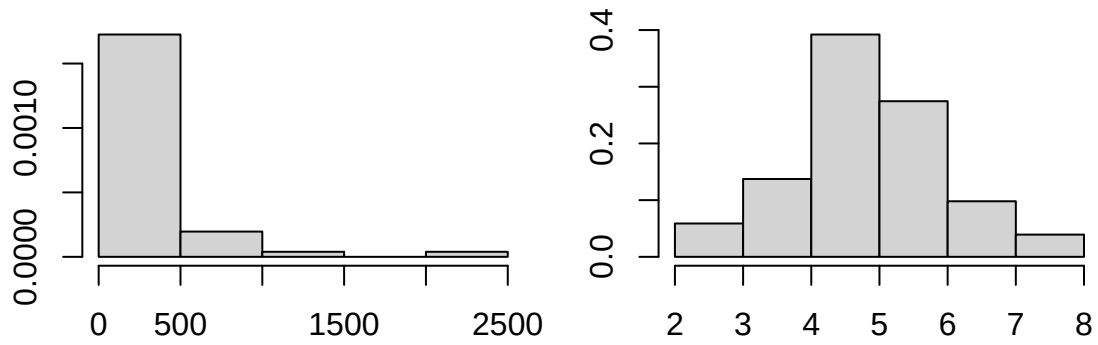


Figure 5.26: Histograms of raw liver transplant waiting time data (left panel) and logs of the data (right panel).

After taking logarithms, the distribution appears to be very close to normal. Figure 5.27 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.

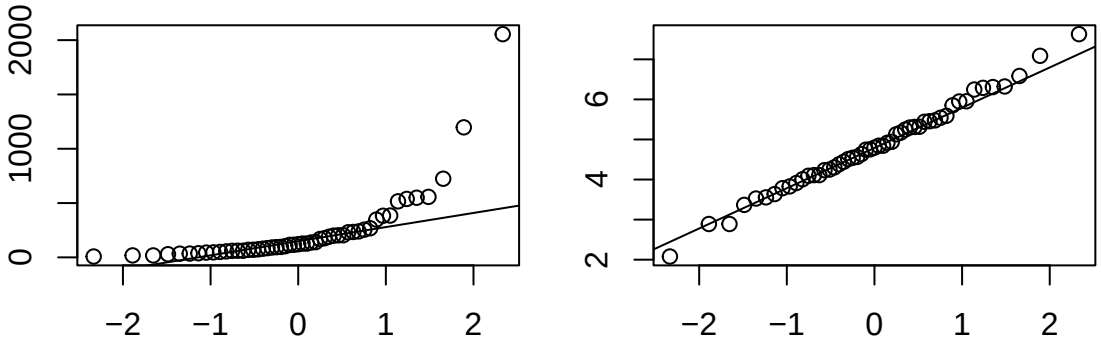


Figure 5.27: Normal QQ-plots of raw liver transplant waiting time data (left panel) and logs of the data (right panel).

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))
```

5.4.3 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:

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}dx. \tag{5.7}$$

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:

$$\Gamma(\alpha) = (\alpha - 1)\Gamma(\alpha - 1), \quad \text{when } \alpha > 1. \tag{5.8}$$

This follows from application of one step of integration by parts to the integral in equation (5.7).

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 (5.8). We find that $E[X] = \alpha$, and $E[X^2] = \alpha(\alpha - 1)$. The variance can then be obtained as $\text{Var}(X) = \alpha$.

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 5.44 *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 5.45 *Repeating the final simulation experiment of Example 5.44 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.

5.5 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 5.46 *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 5.47 *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.

5.5.1 *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 5.48 *Returning to Example 5.47, we can estimate the standard error of the estimate of I:*

```
standev <- sd(integrand)
se <- 2*standev/sqrt(n)
se

## [1] 0.00369
```

5.5.2 *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 5.49 *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.

5.5.3 *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 5.50 *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).

5.6 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 5.49 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

$$\mathrm{Var}((Y+Z)/2) = \mathrm{Var}(Y)/4 + \mathrm{Var}(Z)/4 + \mathrm{Cov}(Y,Z)/2.$$

If Y and Z have the same distribution, this simplifies to

$$\mathrm{Var}((Y+Z)/2) = \mathrm{Var}(Y)/2 + \mathrm{Cov}(Y,Z)/2,$$

and the covariance is certain to be no more than the variance of Y. This means that, in general,

$$\mathrm{Var}((Y+Z)/2) \leq \mathrm{Var}(Y).$$

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 5.49, 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
```

5.7 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 5.28. 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)
```

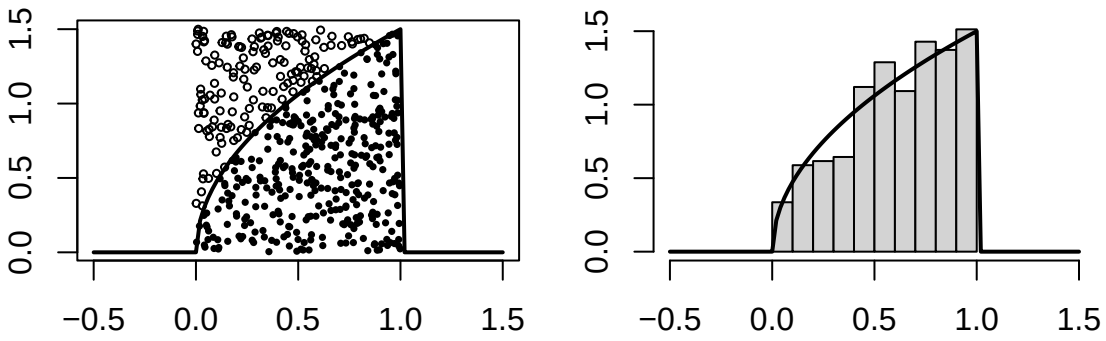


Figure 5.28: Illustration of rejection sampling concept. The solid black corresponds to the target PDF from which we wish to simulate random numbers. 500 pairs of uniform random variates are plotted in the left panel. The horizontal coordinates of those that are under the PDF curve in the left panel 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 5.51 *Consider the biweight probability density function, plotted in Figure 5.29:*

$$f_X(x) = \begin{cases} \frac{15}{16}(1-x^2)^2, & \textit{for } x \in [-1,1] \\ 0, & \textit{otherwise} \end{cases}$$

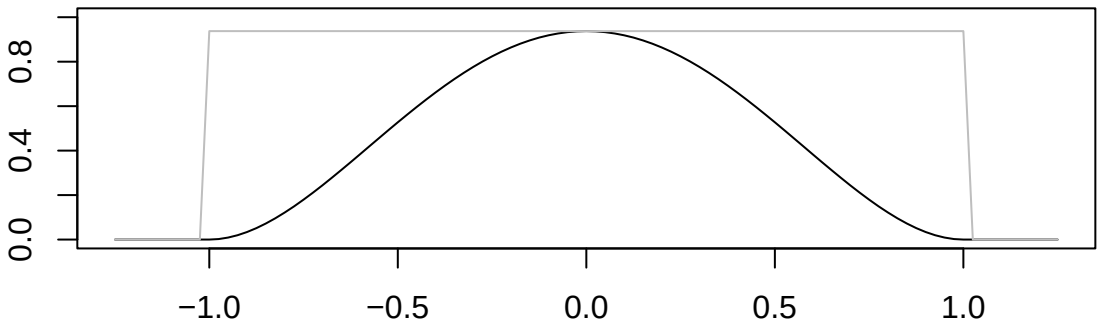


Figure 5.29: PDF of the biweight distribution.

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
```

5.7.1 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 5.52 *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 5.30.

5.7.2 The normalization constant is not required information

The following function can be used to generate `n` pseudorandom numbers from the same distribution as above.

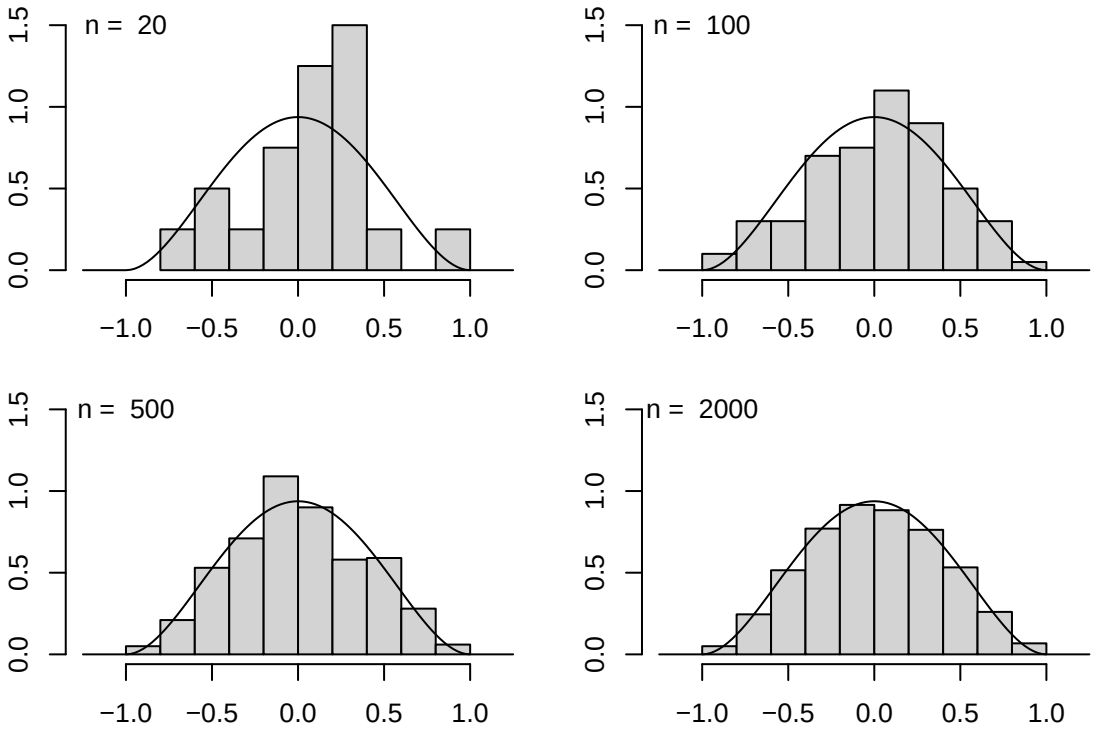


Figure 5.30: Histograms of samples from biweight distribution with samples of size 20, 50, 100, and 200, with overlaid PDF curve.

```
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 5.53 *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.*

5.7.3 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 5.31 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.

5.7.4 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 5.54 *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

$$F_X(x) = P(X \leq x) = \begin{cases} 0 & x \leq 0 \\ 0.5x^3 & 0 < x \leq 1 \\ 0.5x & 1 < x \leq 2 \\ 1 & x > 2 \end{cases}$$

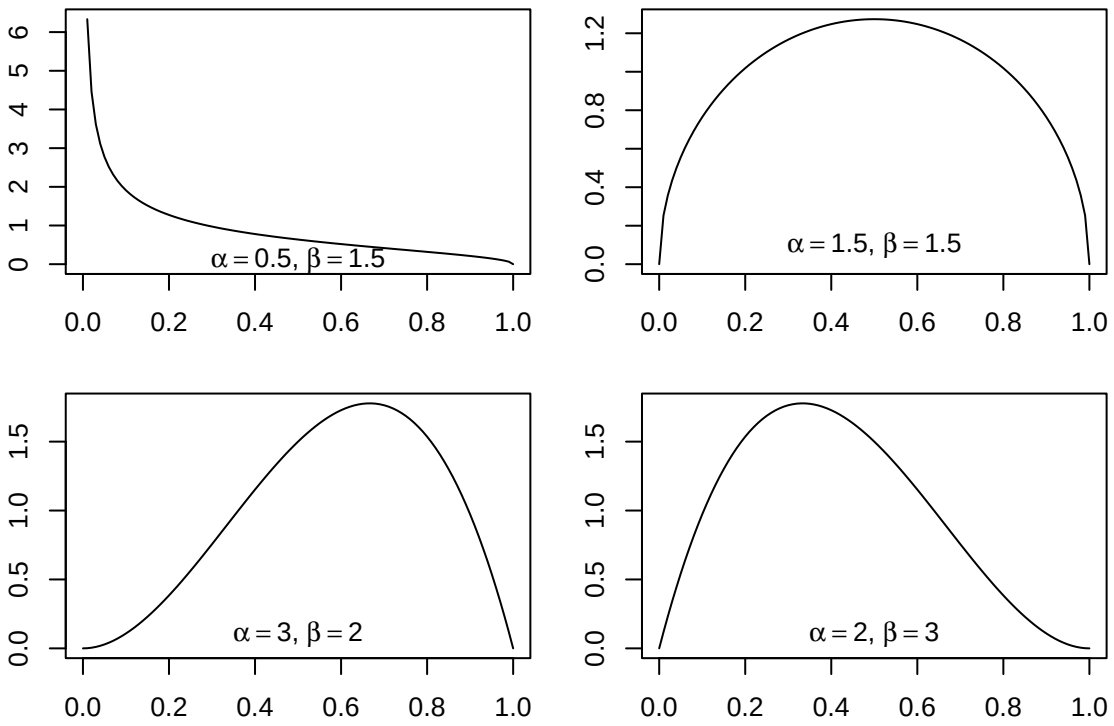


Figure 5.31: Various beta PDF curves.

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
```

5.7.5 *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 5.55 *Figure 5.32 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*

$$f_X(x) = \begin{cases} 2 - 2|1 - 2x|, & 0 \leq x < 1 \\ 0, & \textit{otherwise} \end{cases}$$

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 5.33 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)
```

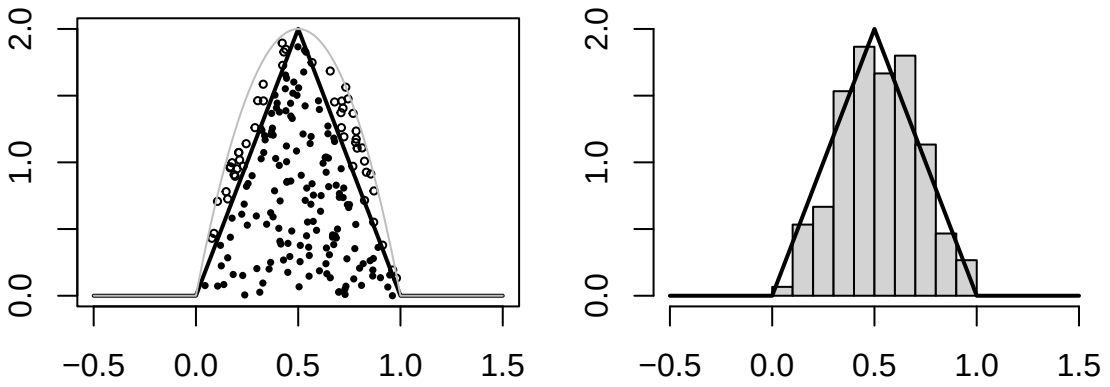


Figure 5.32: Illustration of rejection sampling concept. The solid black corresponds to the target PDF from which we wish to simulate random numbers. 200 pairs of beta random variates are plotted in the left panel. The proposal density (multiplied by 4/3) is plotted in grey. The horizontal coordinates 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.

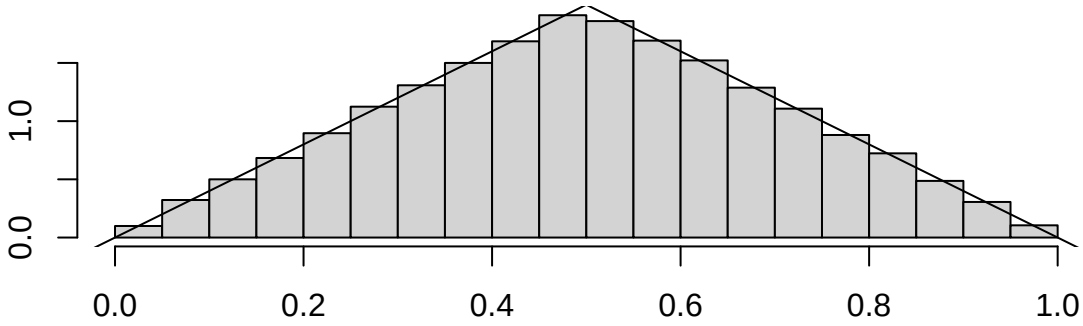


Figure 5.33: Histogram of random variates from the triangular density generated by rejection sampling using a beta proposal density.

5.7.6 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 5.56 *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 5.34 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 5.34 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
```

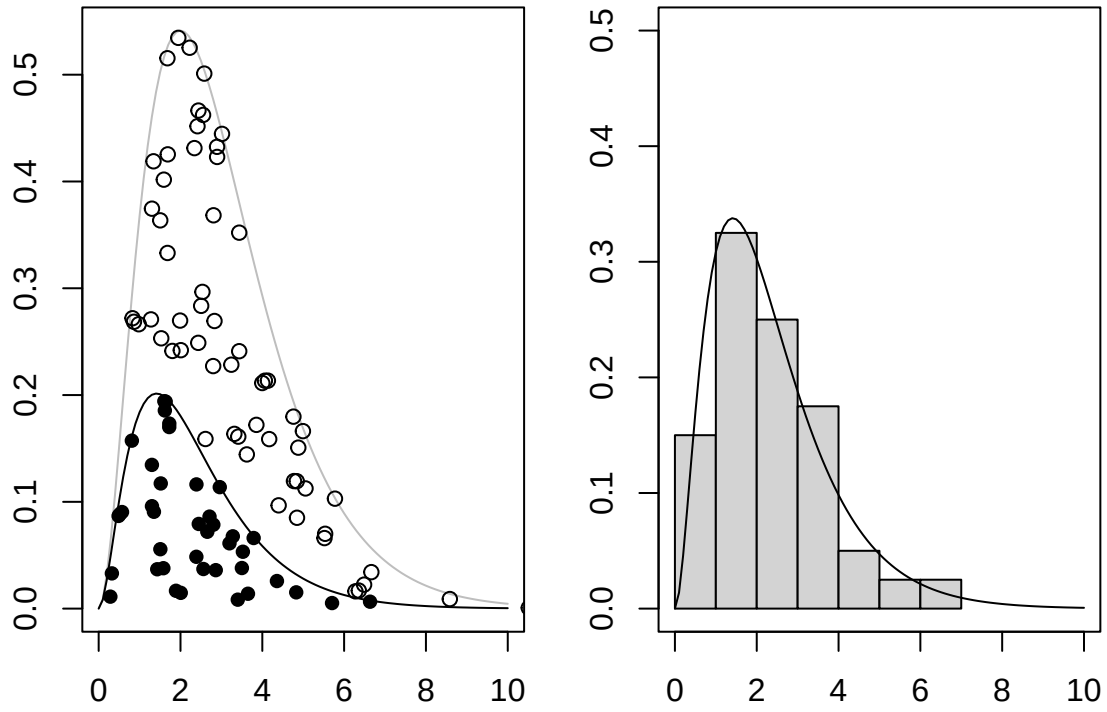


Figure 5.34: Left panel: 100 proposed values from gamma distribution (grey curve) with accepted values for unnormalized target PDF (black curve) of Example 5.56; Right panel: histogram of accepted values with overlaid normalized PDF curve.

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 $\int_0^5 \sin(\mathrm{e}^x)dx$ to obtain an integral of the form $\int_0^1 g(u)du$. Then apply antithetic sampling to obtain an estimate of this integral.

6

Modelling Several Independent Random Variables

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. Many modelling options become available when 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. We will develop some tools in this chapter to build up these more complex models from relatively simple building blocks, using the concept of statistical independence.

6.1 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 6.1 *Suppose x_1 and x_2 are random variables governed by the joint PDF*

$$f_{X_1,X_2}(x_1, x_2) = x_1 + 2x_3^2, \quad \textit{for } 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1,$$

and 0, otherwise. A contour plot of this function is displayed in Figure 6.1, together with a perspective plot.

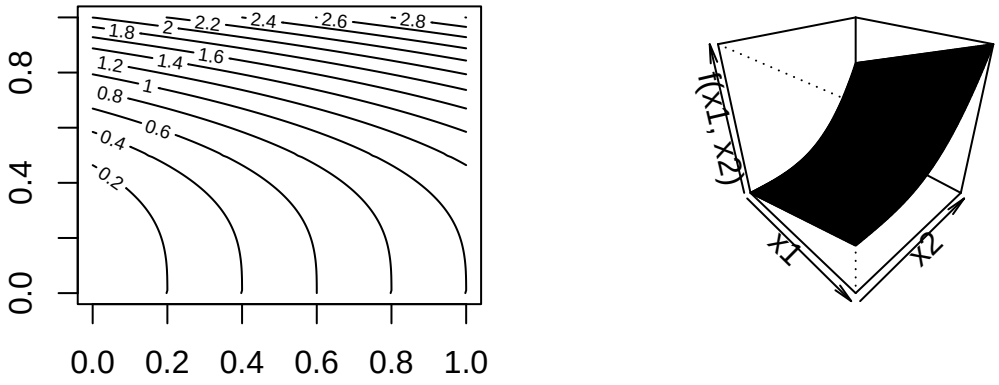


Figure 6.1: Contour plot of joint PDF described in Example 6.1.

Example 6.2 *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 6.2.

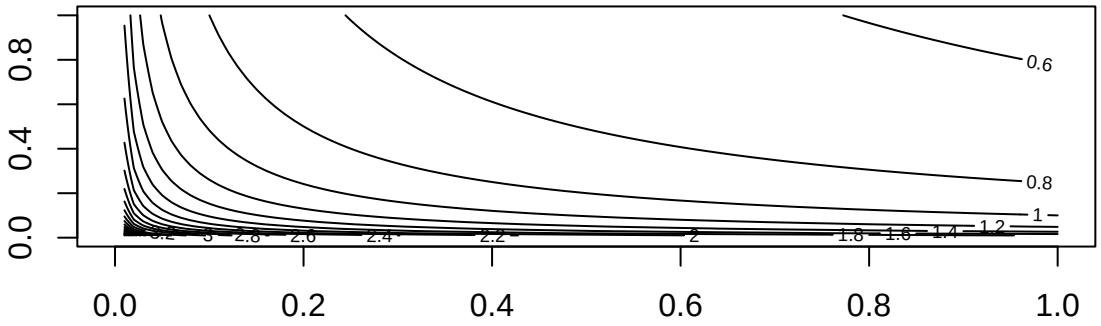


Figure 6.2: Contour plot of joint PDF described in Example 6.2.

In this case, the probability density decreases as x_1 and x_2 jointly increase.

6.1.1 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 6.3 *Returning to Example 6.2, 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 6.4 *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:*

$$f_{X,T}(x,t) = \frac{x + \frac{t}{5} - 13}{5}, \ 10 \le x \le 11, \ 15 \le t \le 20$$

with $f_{X,T}(x,t) = 0$ for other values of x and t.

6.2 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,

$$E[X_1] = \int_0^1 \int_0^1 x_1(\alpha+1)^2(x_1x_2)^\alpha dx_1 dx_2 = \frac{\alpha+1}{\alpha+2}.$$

An identical calculation shows that $E[X_2] = \frac{\alpha+1}{\alpha+2}$. Also,

$$E[X_1X_2] = \int_0^1 \int_0^1 x_1x_2(\alpha+1)^2(x_1x_2)^\alpha dx_1 dx_2 = \frac{(\alpha+1)^2}{(\alpha+2)^2}.$$

Example 6.5 *In Example 6.4, the pressure in the cylinder is proportional to XT. Suppose the relation is*

$$P = 3XT$$

Find $E[P]$.

$$\begin{aligned} E[P] = E[3XT] &= \int_{10}^{11} \int_{15}^{20} 3xt \frac{x + \frac{t}{5} - 13}{5} dt dx \\ &= \int_{10}^{11} \frac{105x^2 - 995x}{2} dx \\ &= 568.75 \end{aligned}$$

The covariance of two random variables gives a measure of their association, and can be calculated using

$$\text{Cov}(X_1, X_2) = E[X_1X_2] - E[X_1]E[X_2].$$

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

$$f_{X_1,X_2}(x_1, x_2) = \frac{\lambda}{x_1} e^{-\lambda x_1 - x_2/x_1}, \quad x_1, x_2 \geq 0$$

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

$$E[X_1X_2] = \lambda \int_0^\infty \int_0^\infty \frac{x_1x_2}{x_1} e^{-\lambda x_1 - x_2/x_1} dx_1 dx_2 = \frac{2}{\lambda^2}.$$

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.

6.2.1 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 6.6 *Brain and body weights for a sample of mice are recorded in the data set* `litters` *which can be found in the DAAG library.*

```
library(DAAG)    # load DAAG (after
  #  install DAAG 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 6.3 which is produced with the following code:

```
plot(brainwt ~ bodywt, data = litters)
```

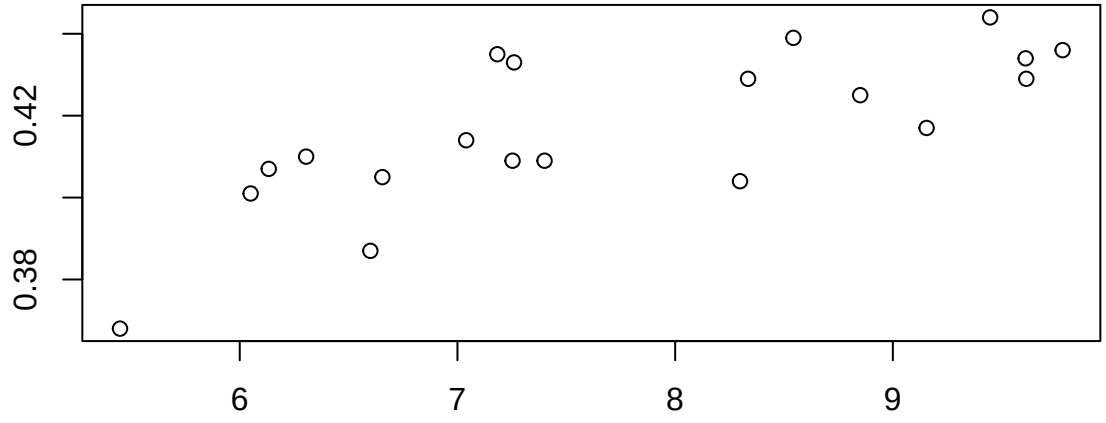


Figure 6.3: Graphical view of positively correlated data: the litters data of Example 6.6.

A best-fit line passing through these data would have a positive slope.

6.3 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

$$f_{X_1}(x_1) = \int_0^1 (\alpha+1)^2 (x_1 x_2)^\alpha dx_2 = (\alpha+1) x_1^\alpha.$$

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

$$f_{X_1,X_2}(x_1,x_2) = \frac{\lambda}{x_1} e^{-\lambda x_1 - x_2/x_1}, \quad x_1, x_2 \geq 0$$

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.

6.4 Independence

Random variables X_1 and X_2 are said to be independent when X_1 provides no predictive information about X_2, and vice versa.

Independence is actually a property of the joint distribution. Specifically, random variables are independent if their joint distribution can be factored into a product of univariate marginal probability density functions. That is, X_1 and X_2 are independent, if their joint density is

$$f_{X_1,X_2}(x_1,x_2) = f_{X_1}(x_1) f_{X_2}(x_2) \tag{6.1}$$

where the two functions are the respective marginal density functions of X_1 and X_2.

6.5 Modelling sequences of random variables

When dealing with more than two random variables, the concepts of joint PDF, marginal 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.
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.

6.6 General results concerning collections of measurements

6.6.1 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 6.7 *We reconsider Example 6.4. 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 6.8 *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 6.9 *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

$$E[\overline{X}] = \mu = 15.$$

6.6.2 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

$$\text{Var}(X_1 + X_2 + \cdots + X_m) = m\sigma^2$$

and

$$\text{Var}(\overline{X}) = \sigma^2/m$$

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 6.10 *Reconsider Example 6.9. 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,

$$Var(\overline{X}) = 100/6 = 16.7.$$

6.7 Multiple integration

[1]

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 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

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2} g(x_1, x_2) dx_1 dx_2$$

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 6.11 *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] 0.5009465
```

Calculating the standard error of this estimate is left as an exercise.

[1] This topic may be omitted.

6.8 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.

6.8.1 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 6.12 *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 6.4, and the maximum appears to be near 0.2.

Calculus can often, though not always, be employed to obtain the maximum value, the so-called maximum likelihood estimator. In Example 6.12, 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 $\widehat{\lambda} = 0.2$.

6.8.2 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.

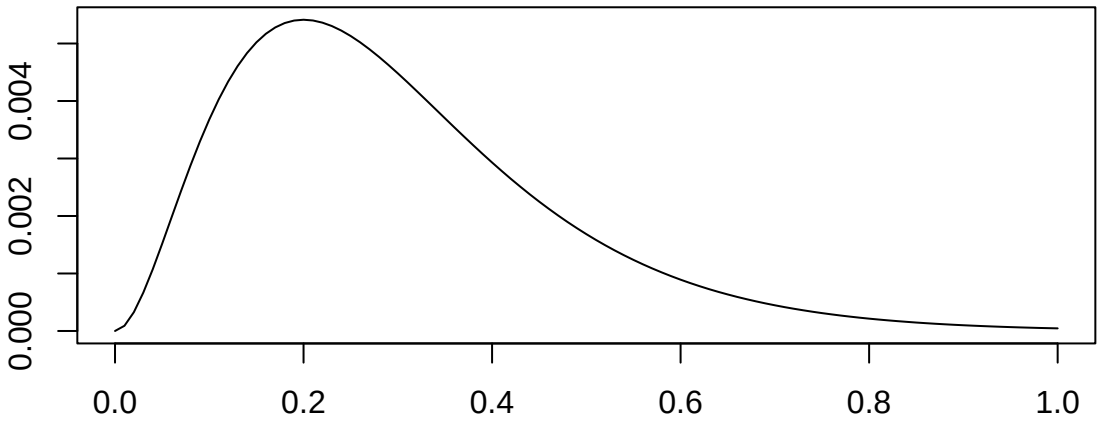


Figure 6.4: Likelihood function for Example 6.12.

Example 6.13 *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 $\widehat{\sigma}^2 = 122$

Exercises

1. Refer to Example 6.4.

   (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 6.11 and use this in a 95% confidence interval.

7. Refer to Example 6.12. 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 6.13. 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

$$f(\varepsilon)=\frac{e^{-(\varepsilon)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}}$$
$$=\frac{e^{-(W_i-gm_i)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}}$$

so the likelihood for the data is

$$L(g,\sigma^2)=\frac{e^{-\sum_{i=1}^{4}(W_i-gm_i)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}^4}$$

Find the maximum likelihood estimates for g and σ^2.

7

Regression and Time Series Models

In this chapter, we bring together the building blocks developed in the previous chapters to construct and simulate from statistical models that are used to make predictions and forecasts. Regression models are predictive models that exploit relationships among two or more variables, and time series models exploit relationships that arise in sequences of observations. We will briefly consider simple regression models - which are relatively straightforward - before turning to a few of the many possibilities that exist for time series.

7.1 Modelling and simulating simple linear regression

In Chapter 2, we saw how information about one variable could be used to make predictions on another variable, through the use of simple regression. There, we used salinity concentration of the soil, x, to predict tomato crop yield, $\mathbf{Y}$. Such a predictive model assumes that the measurement of $\mathbf{Y}$ (called the response) changes in a linear fashion with a setting of the variable $\mathbf{x}$ (called the predictor):

$$\underset{\text{response}}{\mathbf{Y}} = \underset{\text{linear relation}}{\beta_0 + \beta_1 \mathbf{x}} + \underset{\text{noise}}{\varepsilon}$$

- The linear relation is deterministic or non-random.
- The noise or error is thought of as random.

The noise accounts for the variability of the observations about the straight line. If there is no noise, then the relation is deterministic. Increased amounts of noise contribute to increased variability. The level of variability is usually related to the standard deviation, and often, a normal distribution is used to model the noise distribution. The mean of this distribution is taken to be 0.

Let us simulate n observations from the model

$$\mathbf{Y} = 3 + 4x + \varepsilon.$$

We can simulate n noise values from code such as

```
epsilon <- rnorm(n, sd = sigma)
```

where `n` and `sigma` are taken as input.
For example, if $n = 11$ and $\sigma = 3.0$, we can simulate noise using

```
epsilon <- rnorm(11, sd = 3)
```

The predictor values can arise in one of two ways. Either

1. the $\mathbf{x}$'s are fixed values and measured without error (controlled experiment)

   OR

2. the analysis is conditional on the observed values of $\mathbf{x}$ (observational study).

In the first case, the $\mathbf{x}$ values might be a sequence of equally spaced values such as

```
x <- seq(0, 2, length = 11)
```

Then we can simulate the responses from

```
Y <- 3 + 4*x + epsilon
```

The simulated data are

```
data.frame(x, Y)
```

```
##       x         Y
## 1   0.0  4.756586
## 2   0.2  5.928398
## 3   0.4  4.272090
## 4   0.6  4.039508
## 5   0.8  8.017662
## 6   1.0  1.546132
## 7   1.2  9.690296
## 8   1.4  7.771448
## 9   1.6  8.547521
## 10  1.8  7.442034
## 11  2.0 10.651257
```

In the second case, the $\mathbf{x}$ values could arise from any kind of random distribution, so we might think of normally distributed values or uniform values, for example. In this case, we can generate $\mathbf{x}$ from the uniform distribution on $[0, 2]$, and $\mathbf{Y}$ from

```
x <- runif(11, min = 0, max = 2)
Y <- 3 + 4*x + epsilon
```

Now, the simulated data are

```
data.frame(x, Y)
```

```
##             x         Y
## 1  1.93083065 12.479909
## 2  1.41496375 10.788253
## 3  1.28908527  7.828431
## 4  0.77965697  4.758136
## 5  1.39708728 10.406011
## 6  1.08811573  1.898595
## 7  0.45293436  6.702033
## 8  0.96911551  6.047910
## 9  1.58601434  8.491578
## 10 0.01197526  0.289935
## 11 0.37542489  4.152956
```

Exercises

1. Experiment with this simulation program, using the default arguments. Then simulate data from the model $\mathbf{Y} = 2 - 4x + \varepsilon$, where the $\mathbf{x}$ values are uniformly distributed on the interval $[0, 5]$ and $\sigma = 0.5$.

```
simple.sim <- function(intercept=0,slope=1,x=seq(1,10),sigma=1){
    noise <- rnorm(length(x),sd = sigma)
    y <- intercept + slope*x + noise
    title1 <- paste("sigma = ", sigma)
    plot(x,y,pch=16,main=title1)
    abline(intercept,slope,col=4,lwd=2)
}
```

7.2 Autoregressive time series models

Suppose ε_1 and Z_0 are independent normal random variables, and suppose α is a constant. Suppose the expected value of ε_1 is 0, and the variance is $\sigma_\varepsilon^2 > 0$. Let

$$Z_1 = \alpha Z_0 + \varepsilon_1.$$

It is possible to show that Z_1 is normally distributed, and using the results of this section on expected value and variance, Z has mean $\alpha\mu_{Z_0}$ and variance $\sigma_{Z_0}^2\alpha^2 + \sigma_\varepsilon^2$.

We now want to find conditions on α and μ_{Z_0} so that the distribution of Z_1 will be exactly the same as the distribution of Z_0. This kind of stationarity condition is often useful in modelling of processes that occur in time (or in space, for that matter).

$$E[Z_0] = \mu_{Z_0} = E[Z_1] = \alpha\mu_{Z_0}$$

implies that either $\alpha = 1$ or $\mu_{Z_0} = 0$.

$$V(Z_0) = \sigma_{Z_0}^2 = V(Z_1) = \sigma_\varepsilon^2 + \alpha^2\sigma_{Z_0}^2.$$

If $\alpha = 1$, then $\sigma_\varepsilon = 0$, which is not possible. Therefore, $\alpha \neq 1$. This means $\mu_{Z_0} = 0$.

But we also have

$$\sigma_{Z_0}^2 = \sigma_\varepsilon^2 + \sigma_{Z_0}^2\alpha^2$$

so that

$$\sigma_{Z_0}^2(1 - \alpha^2) = \sigma_\varepsilon^2$$

which implies that $\alpha^2 < 1$, and

$$\sigma_{Z_0}^2 = \frac{\sigma_\varepsilon^2}{1 - \alpha^2}.$$

Summarizing the results of the example, we observe that if Z_0 has a normal distribution with mean 0 and variance $\frac{\sigma_\varepsilon^2}{1-\alpha^2}$, independent of ε_1 which also has a normal distribution with mean 0 and variance σ_ε^2, then

$$Z_1 = \alpha Z_0 + \varepsilon_1$$

has the same distribution as Z_0.

Now, let ε_2 be another normal random variable, independent of ε_1 but with the same mean and variance. Then

$$Z_2 = \alpha Z_1 + \varepsilon_2$$

must have the same distribution as Z_1. In fact, for $n = 2, 3, \ldots$,

$$Z_n = \alpha Z_{n-1} + \varepsilon_n$$

defines a sequence of normal random variables all having mean 0 and variance $\frac{\sigma_\varepsilon^2}{1-\alpha^2}$, when $\alpha^2 < 1$ and the ε's are independent of each other.

Usually, the expected value or mean level of the process is some nonzero value μ, so this value is usually subtracted from the time series. That is, $Z_n = Y_n - \mu$, where Y_n is the original time series, having expected value $E[Y_n] = \mu$. Writing the autoregressive model in terms of Y's, we have

$$Y_n = \mu + \alpha(Y_{n-1} - \mu) + \varepsilon_n.$$

7.2.1 The Markov Property

Recall that if a sequence of random variables $Z_1, Z_2, \ldots, Z_n$ is independent, then their joint distribution can be factored:

$$f(z_1, z_2, \ldots, z_n) = f(z_1)f(z_2)\cdots f(z_n).$$

If the sequence is completely dependent, the factorization involves complicated conditional distributions along the lines of:

$$f(z_1, z_2, \ldots, z_n) = f(z_n|z_1, z_2, \ldots, z_{n-1})f(z_{n-1}|z_1, \ldots, z_{n-2})\cdots$$

$$\cdots f(z_4|z_1, z_2, z_3)f(z_3|z_1, z_2)f(z_2|z_1)f(z_1).$$

Back in the 19th century, A. Markov realized that there might be a useful class of models in between these two extremes.

The function $f(z_3|z_1, z_2)$ can be approximated by $f(z_3|z_2)$. This will usually be a better approximation than $f(z_3)$ only - which is what independence implies. This approximation says that z_3 is dependent on z_1 only through z_2 explicitly. In other words, given z_2, no additional information in the sample will provide information about z_3. Similarly, approximate $f(z_4|z_1, z_2, z_3)$ by the simpler $f(z_4|z_3)$, and so on, finally approximating $f(z_n|z_1, \ldots, z_{n-1})$ by $f(z_n|z_{n-1})$.

That is, assume

$$f(z_1, z_2, \ldots, z_n) = f(z_n|z_{n-1})f(z_{n-1}|z_{n-2})\cdots$$

$$\cdots f(z_4|z_3)f(z_3|z_2)f(z_2|z_1)f(z_1).$$

... This is the Markov property.

7.2.2 Simulating from an AR(1) model

We illustrate how to simulate autoregressive data with an artificial example. Suppose $\phi = -.7$ and $\sigma_\varepsilon^2 = 0.1$. Then $\sigma_{Z_0}^2 = \frac{0.1}{1-(-.7)^2} = 0.196$. Therefore, we can simulate Z_0 as a normal random variable with mean 0 and variance 0.196:

```
Z0 <- rnorm(1, sd = sqrt(.196))
Z0

## [1] 0.2093318
```

With Z_0 in hand, we can simulate Z_1 as $\phi_1 Z_0 + \varepsilon_1$,

```
phi1 <- -0.7
eps1 <- rnorm(1, sd = sqrt(.1))
Z1 <- phi1*Z0 + eps1
Z1

## [1] -0.2585139
```

With Z_1, we can simulate Z_2 as $\phi_1 Z_1 + \varepsilon_2$:

```
eps2 <- rnorm(1, sd = sqrt(.1))
Z2 <- phi1*Z1 + eps2
Z2

## [1] 0.5938248
```

and Z_3

```
eps3 <- rnorm(1, sd = sqrt(.1))
Z3 <- phi1*Z2 + eps3
Z3

## [1] -0.7659103
```

and so on. For this kind of calculation, we cannot avoid the use of a `for()` loop.

Set up a vector which has enough elements to store the result

```
n <- 20 # we will simulate 20 values
Z <- numeric(n)
```

Generate enough normal errors, ε, with mean 0 and variance σ_ε^2:

```
eps <- rnorm(n, sd = sqrt(.1))
```

Using a `for()` loop for $i = 1, \ldots, n$, repeatedly overwrite Z_0 with the value of $\phi_1 Z_0 + \varepsilon$, and store the result in successive entries of Z:

```
for (i in 1:n) {
    Z0 <- phi1*Z0 + eps[i]
    Z[i] <- Z0
}
Z # the results are stored here

##  [1] -0.20295182  0.37910121 -0.41218811 -0.25282241
##  [5]  0.27819106  0.10364630 -0.32425944  0.22637999
##  [9] -0.60207497  0.19460789 -0.34095356  0.45885106
## [13] -0.56650993  0.06807187  0.46185246 -0.31891019
## [17]  0.08917575  0.03330476  0.10278182 -0.39956312
```

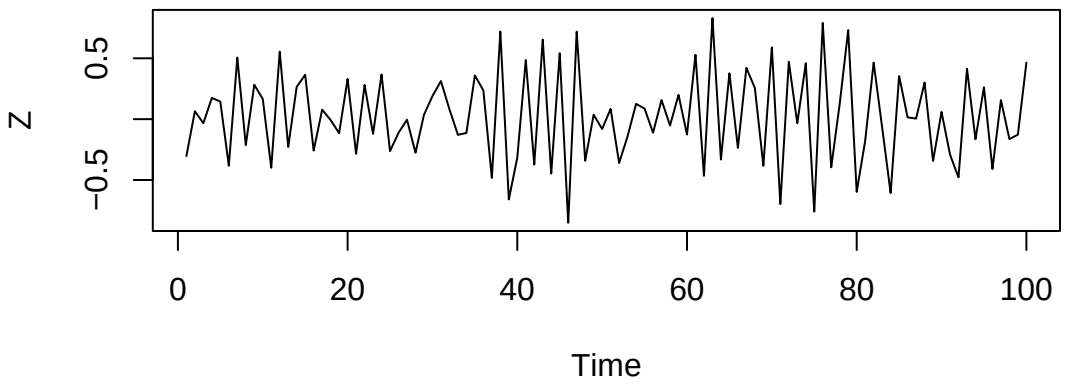
A better way to simulate from an AR model is to use the `arima.sim()` function. In the following, we simulate a time series of 100 observations from the AR(1) process with $\phi = -0.7$:

```
Z <- arima.sim(100, model = list(ar = phi1), sd = sqrt(0.1))
```

7.2.3 The Trace Plot for Time Series Data

The trace plot is just a graph of the time series measurements against time.

```
ts.plot(Z)
```



Observe that the observations are not completely random. They appear to oscillate: a large positive value is often followed by a large negative value.

7.2.4 The lag plot for time series data

The lag plot is just a graph of the successive time series measurements against values at previous lags.

```
lag.plot(Z, lags=4,  do.lines=FALSE)
```

In Figure 7.1, we are looking at the first 4 lag plots. Note how the relation at lag 1 is negative, and positive at lag 2, and negative again at lag 3, and so on.

7.2.5 The ACF plot for time series data

We have seen the ACF plot in our checks for random number generator quality. Figure 7.2 shows a typical ACF plot for an AR(1) process with negative autoregressive parameter:

```
acf(Z)
```

Note how the values oscillate between negative and positive. The values of the ACF at each lag are proportional to the slopes observed in the corresponding lag plots.

7.2.6 The lag 1 autocovariance for AR(1) data

We saw the use of autocorrelations and autocovariances in detecting (linear) dependence problems in random number generation. These tools are helpful in the study of linear time series models.

Recall that the covariance of X and Y is

$$E[XY] - E[X]E[Y].$$

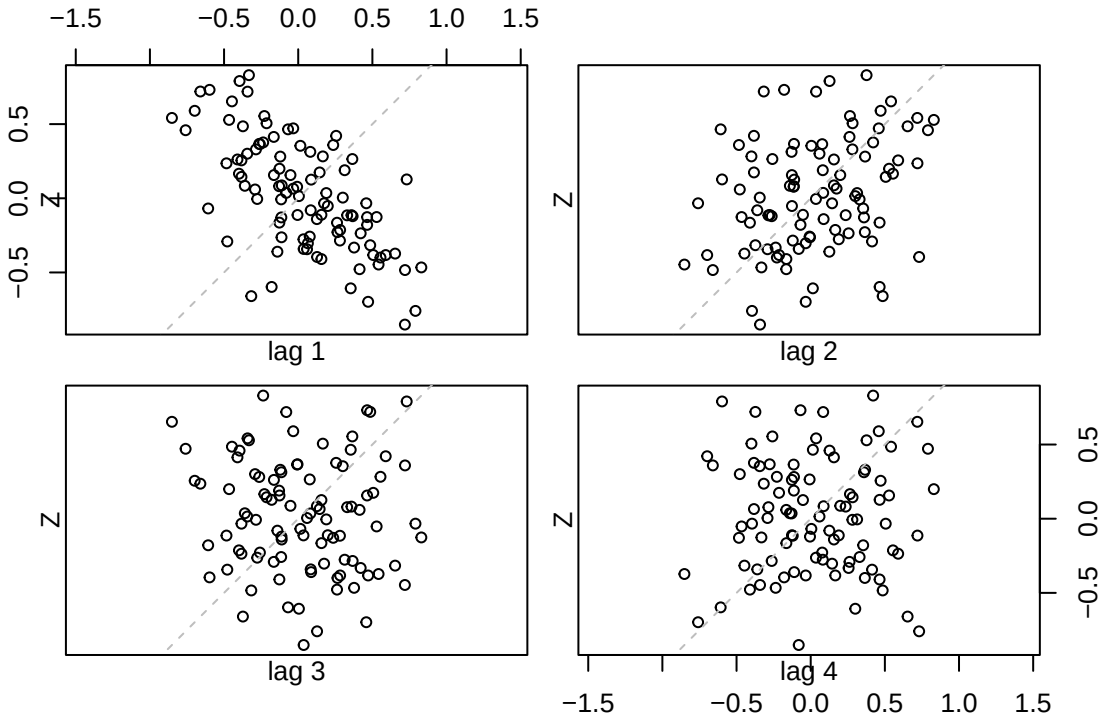


Figure 7.1: The first four lag plots for data simulated from an autoregressive order 1 process.

Since

$$E[Z_n] = 0$$

the covariance of Z_n and Z_{n-1}, also called the lag 1 autocovariance is

$$\gamma_1 = E[Z_n Z_{n-1}]$$

$$= E[(\phi_1 Z_{n-1} + \varepsilon_n) Z_{n-1}] =$$

$$= \phi_1 E[Z_{n-1}^2] + 0$$

$$= \phi_1 \sigma_Z^2.$$

Recall that the correlation is defined as the covariance of X and Y divided by the product of the standard deviations of X and Y. For a stationary AR(1) process, the standard deviations of Z_{n-1} and Z_n are the same, so we divide by the variance of Z_n to get the lag 1 autocorrelation:

$$\rho_1 = \frac{\gamma_1}{\text{Var}(Z_n)} = \phi_1.$$

Similar reasoning leads to

$$\rho_k = \phi_1^k$$

7.2.7 The theoretical autocorrelation function for the AR(1) process

Here is a plot of the autocorrelation function for an AR(1) process with $\phi_1 = .8$:

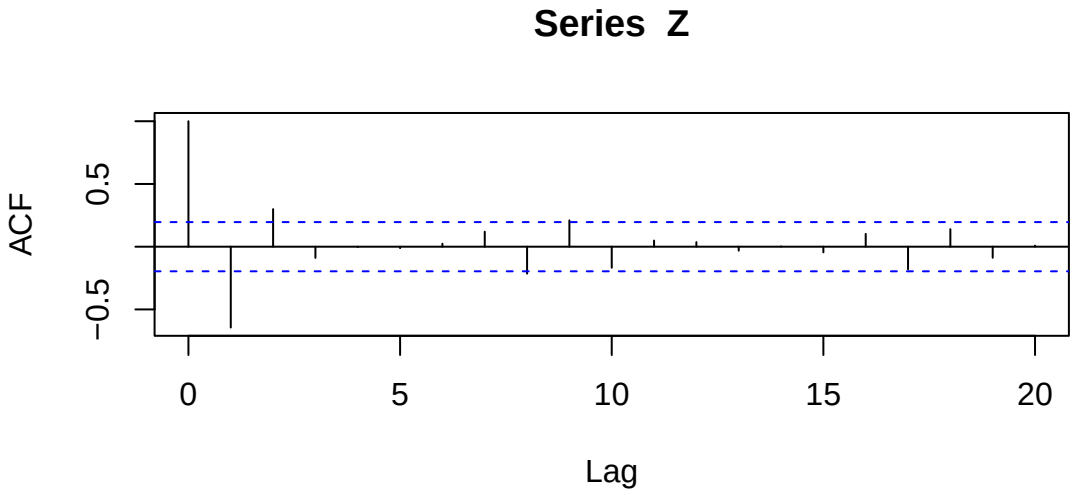
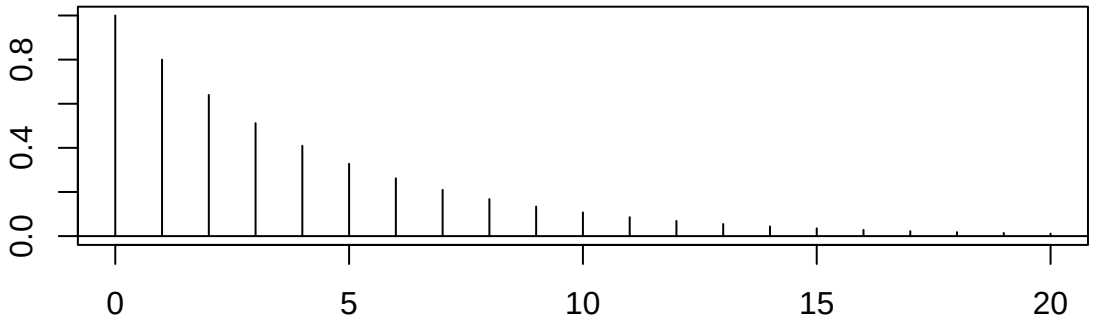
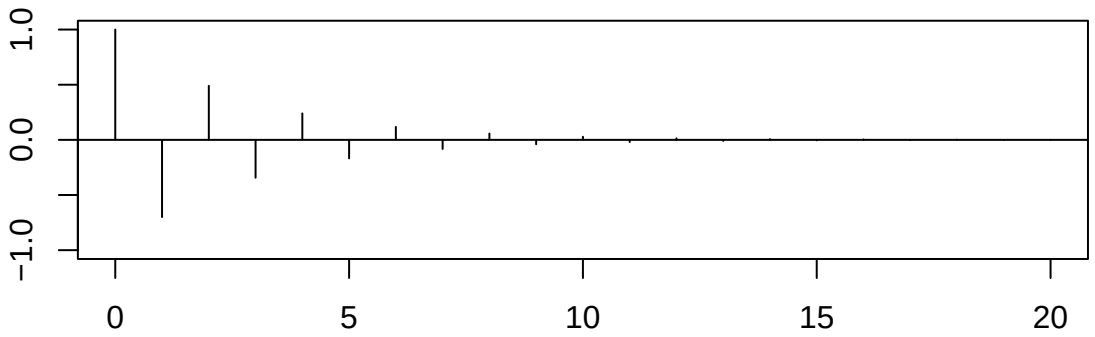


Figure 7.2: ACF plot for data simulated from an autoregressive order 1 process.



All data points are correlated with each other, but dependence decays exponentially with increasing time between the points.

Here is a plot of the autocorrelation function for an AR(1) process with $\phi_1 = -.7$:



Again, the correlations are all nonzero, but this time each observation is negatively correlated with the

previous observation.

7.2.8 An autoregressive time series model with nonzero mean

Usually, the expected value or mean level of the process is some nonzero value μ, so this value is usually subtracted from the time series. That is, $Z_n = Y_n - \mu$, where Y_n is the original time series, having expected value $E[Y_n] = \mu$. Writing the autoregressive model in terms of Y's, we have

$$Y_n = \mu + \phi_1(Y_{n-1} - \mu) + \varepsilon_n.$$

7.2.9 An example: Lake Huron data

We consider the annual levels of Lake Huron in the `LakeHuron` data object. The trace plot is pictured in Figure 7.3.

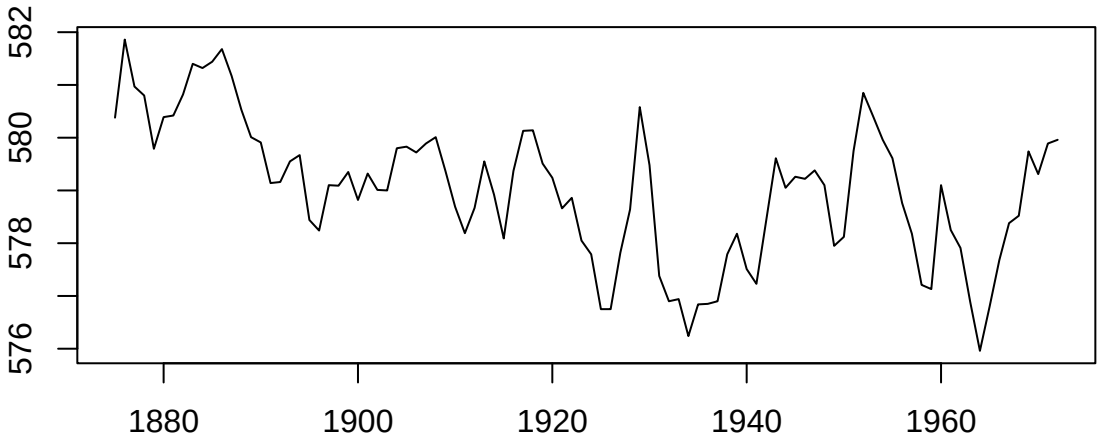


Figure 7.3: Lake Huron level data.

The sample ACF is plotted in Figure 7.4.

```
acf(LakeHuron)
```

The sample ACF looks very similar to the theoretical ACF for an AR(1) process with positive ϕ_1.

Fitting an AR(1) model

The parameters of the AR(1) model can be estimated by maximum likelihood estimation with the arima() function:

```
lh <- arima(LakeHuron, order=c(1,0,0))
lh

##
## Call:
## arima(x = LakeHuron, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
```

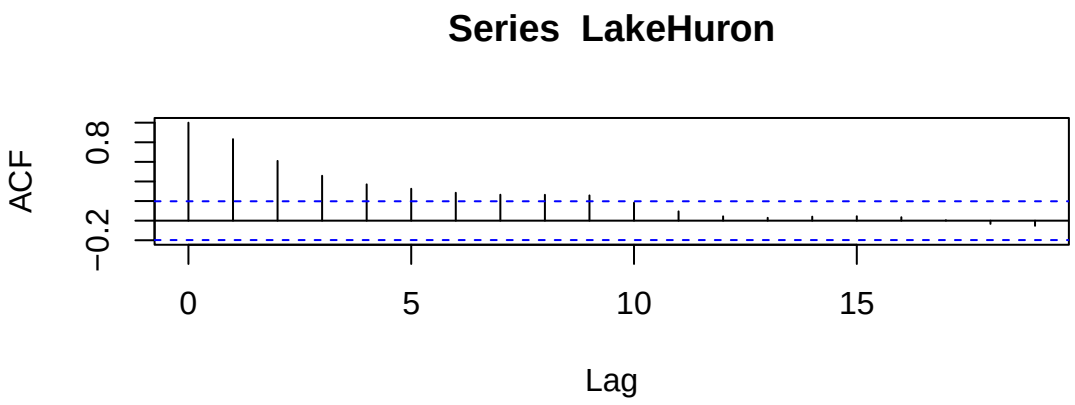


Figure 7.4: Autocorrelation function for Lake Huron level data.

```
##        0.8375   579.1153
## s.e.   0.0538     0.4240
##
## sigma^2 estimated as 0.5093:  log likelihood = -106.6,  aic = 219.2
```

The first component of the order parameter specifies the order of the autoregressive model: 1. The other two components are 0, in this case. We will see their use later.

The output from the function tells us that the ϕ_1 parameter is estimated as 0.8375, and μ is estimated at 579.1153. Standard error estimates are provided for these estimates as well. Both are small relative to the parameter estimates indicating that we have a fair bit of precision.

The variance σ^2 is also estimated as 0.5093.

Predicting with the fitted model

The fitted autoregressive model is

$$\widehat{Y}_n = 579.1 + .8375(Y_{n-1} - 579.1)$$

By plugging in a value for Y_{n-1}, we can use the fitted value to predict Y_n. The variance of the prediction can be calculated easily as well which gives us a standard error for the prediction (when we take a square root). This is all coded in the `predict.Arima()` function. For example, to predict the next 5 years of Lake Huron levels, use

```
predict(lh, n.ahead=5)
```

```
## $pred
## Time Series:
## Start = 1973
## End = 1977
## Frequency = 1
## [1] 579.8228 579.7078 579.6116 579.5310 579.4634
##
## $se
## Time Series:
## Start = 1973
```

```
## End = 1977
## Frequency = 1
## [1] 0.7136434 0.9308795 1.0569470 1.1370689 1.1900573
```

Simulating the AR(1) process mimicking the Lake Huron levels runs as follows:

```
n <- length(LakeHuron); phi1 <- .8375; sigma2 <-.5093;
mu <- 579.1153
Z <- arima.sim(n, model=list(ar=phi1), sd=sqrt(sigma2))
SimLake2 <- Z + mu  # add back the mean level
```

Graphing the Simulated Data

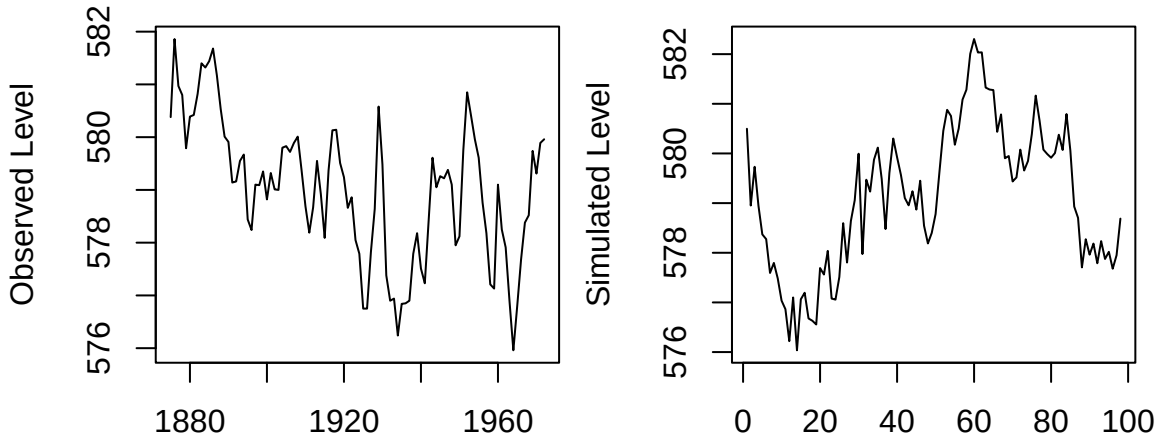


Figure 7.5: Left panel: Lake Huron levels for the years 1875 through 1972. Right panel: 98 years of data simulated from an autoregressive process designed to mimic the behaviour of the Lake Huron levels, arising from the arima.sim function.

7.2.10 Risk assessment via simulation

With a good predictive model, we can repeatedly simulate series of realistic observations, where the dependence structure in the data has been accounted for, to determine, for example, probabilities of extreme behaviour, such as very high levels or very low levels.

Example 7.1 *When lake levels rise above a certain point, flooding occurs, but if flooding has not occurred in the past, models are needed to assist with calculating the probability of a flood in the future. The so-called 100-year flood event corresponds to a water level that is exceeded with probability 0.01 in any given year. This can be estimated from the 99th percentile of the observed lake levels, but when doing such estimation there is considerable uncertainty.*

In this example, we illustrate how the autoregressive model can give more accurate assessments of the uncertainty in 100-year flood risk at Lake Huron than the assumption of independence. Here, we are assuming that the climatic conditions in southern Ontario and southern Michigan are not changing; incorporating climate change effects would be needed to make this model more accurate. We first simulate a large number of samples of 98-year time series, each mimicking the Lake Huron lake levels. In each case, we find the 99th percentile, giving us a set of 1000 such percentiles. We then calculate the standard deviation of these percentiles to obtain an estimated standard error.

```
quantile(LakeHuron, .99) # observed 99th percentile

##      99%
## 581.6854

q99 <- NULL
for (i in 1:1000) {
Z <- arima.sim(98, model=list(ar=phi1), sd=sqrt(sigma2))
SimLake2 <- Z + mu  # add back the mean level
q99 <- c(q99, quantile(SimLake2, .99))
}
sd(q99)

## [1] 0.6766567
```

This means that we would estimate the "100-year-flood" level as 581.7 feet with a standard error of 0.68 feet. Repeating the exercise, assuming that the levels are not dependent from year to year, we first find the standard deviation of the 98 observed values as 1.3182985*. Then, we simulate 1000 sequences of 98-year series of independent lake levels as follows:*

```
Q99 <- NULL
for (i in 1:1000) {
Z <- rnorm(98, mean=579.0041, sd = 1.318299)
Q99 <- c(Q99, quantile(Z, .99))
}
```

We have also computed the 99th percentiles for each year. Based on this simulation, we calculate the standard error of the 99th percentiles as

```
sd(Q99)

## [1] 0.4019099
```

Based on this standard error, we could obtain an approximate 95% confidence interval for the true 99th percentile as 581.7 ± 0.79 *which is a much narrower interval than what we obtain from the more accurate autoregressive analysis:* 581.7 ± 1.33*.*

Using the independence model could give a misleading impression of the true degree of uncertainty to engineers and planners who might be developing flood protection infrastructure.

7.3 Stationary time series

The sequence Z_n is an example of a stationary time series, because its distributions do not change from time to time. Each random variable has the same normal distribution. The sequence Z_n is also an example of a causal time series. The distribution of the jth value of the time series depends only on the previous $j-1$ values, not on the "future" values. Another example of a stationary causal time series is the autoregressive order 2 (AR(2)) process:

$$Z_n = \alpha_1 Z_{n-1} + \alpha_2 Z_{n-2} + \varepsilon_n$$

as long as $|\alpha_2| < 1$, $\alpha_1 + \alpha_2 < 1$ and $\alpha_2 - \alpha_1 < 1$. Including the mean level, the model becomes

$$Y_n = \mu + \alpha_1(Y_{n-1} - \mu) + \alpha_2(Y_{n-2} - \mu) + \varepsilon_n.$$

Higher order AR processes can be considered as well, where Y_n depends on additional terms.

Example 7.2 AR(2): Annual Canadian Lynx

The data in the `lynx` *object concern the annual numbers of lynx trapped in northern Canada for the years 1821 through 1934. Figure 7.6 contains a trace plot of these counts, produced from the following code:*

```
ts.plot(lynx) # draw a broken line curve through the data points
points(1821:1934, (lynx)) # include the data points on the plot
```

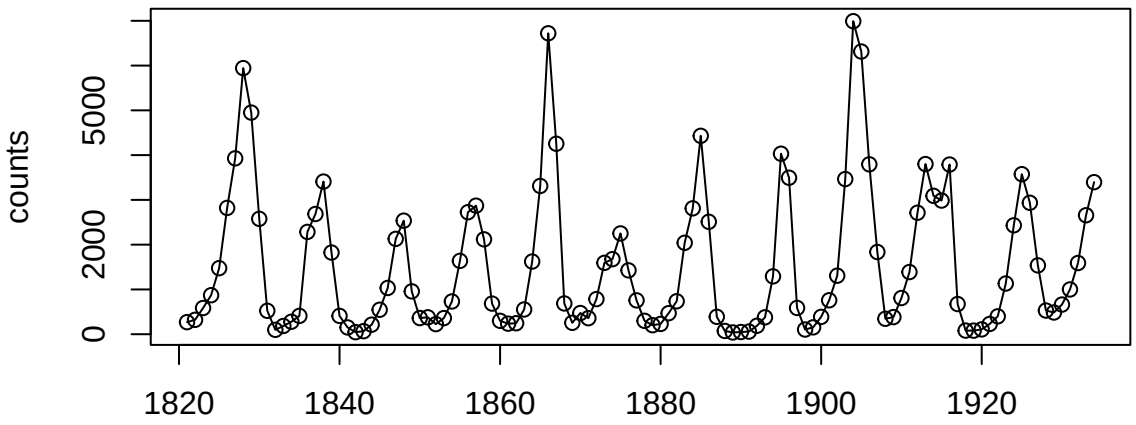


Figure 7.6: The number of lynx trapped in northern Canada annually.

What is obvious on the plot is the periodic behaviour. Every few years the counts dramatically increase before just as dramatically collapsing, almost to 0, remaining at that level for awhile before repeating the cycle. What is also evident on the plot, though not quite as obviously, is that the variability of the counts changes, depending upon the stage of the cycle. Note, in particular, that when the troughs of the curve are at very similar levels, but the peaks of the curve are highly variable. This is an indicator that the variability is not constant.

With count data, it is often a good idea to work with square roots of the counts. This kind of transformation has the effect of substantially reducing very large values while having less effect on small and moderate values: this is a form of variance-stabilizing transformation.

```
ts.plot(sqrt(lynx))
points(1821:1934, sqrt(lynx))
```

Figure 7.7 shows the effect of taking the square root on each of the counts. Now the variability of the troughs is larger while the variability of the peaks is slightly reduced. Overall the variation is about the same, no matter what stage of the cycle one is looking at.

We can fit the AR(2) model using maximum likelihood estimation for the parameters, using the `arima()` *function:*

```
arima(sqrt(lynx), order=c(2,0,0))

##
## Call:
## arima(x = sqrt(lynx), order = c(2, 0, 0))
##
## Coefficients:
##          ar1      ar2  intercept
##       1.3088  -0.7104    34.1280
```

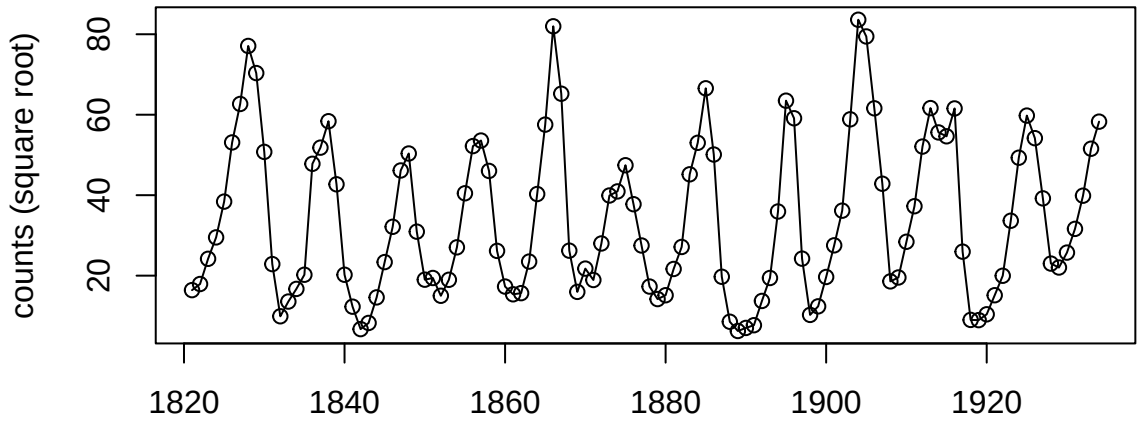


Figure 7.7: The number of lynx trapped in northern Canada annually on the square root scale.

```
## s.e.  0.0648   0.0645     2.0449
##
## sigma^2 estimated as 76.51:  log likelihood = -410.13,  aic = 828.26
```

The fitted model is

$$Z_n = 1.31Z_{n-1} - .7104Z_{n-2} + \varepsilon_n$$

where the error variance is estimated to be 76.51. The mean level was estimated as 34.12, so $Z_n = \sqrt{Y_n} - 34.12$, *where* $Y_j =$ *# of lynx trapped in year* j, *for* $j = 1821, ..., 1934$

We can use simulation to check if an AR(2) model is appropriate for given data. Simulation with a `for()` loop is possible, as it was for the AR(1) model, and the `arima.sim()` function can be used with two values for the `ar` parameter in place of one.

Example 7.3 *Figure 7.8 shows the results of simulating from the AR(2) model for the lynx data studied in Example 7.2. Simulations from the fitted model are compared with real data. The code is below.*

```
par(mfrow=c(2,2));  pars <- c(1.3088, -.7104)
sig <- sqrt(76.51); xbar <- 34.12
for (j in 1:4) {
ts.plot((arima.sim(110, model=list(ar=pars),
       sd=sig)+xbar)^2, ylab="", main=
       paste("Simulation ", j))
}
```

As an approximation, the AR(2) model is a good start, providing more realism than either independence or even an AR(1) model could provide. However, the behaviour of the simulated data tends to be somewhat less regular than the real data, suggesting that an important factor might be missing from the analysis.

7.3.1 Moving Average Processes

Unlike AR processes where the data points are all dependent on each other, the data points of an MA process only depend on each other if they close together in time.

This offers new modelling possibilities.

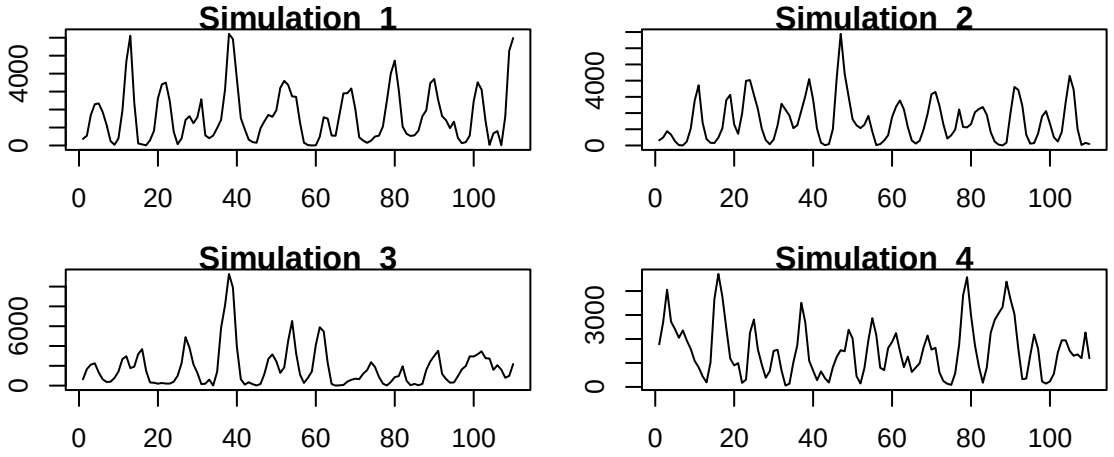


Figure 7.8: Four sets of simulated data from the AR(2) process which was fit to the Canadian lynx time series.

The Moving Average Process of Order 1

The 0-mean moving average process of order 1 (MA(1)) is defined as

$$Z_n = \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

where the ε's are independent and normally distributed with mean 0 and variance σ_2, and θ_1 is a constant.

According to this definition, any ε_k will be independent of Z_j for all $j < k$.

If we define $Y_n = Z_n + \mu$, then Y_n is a MA(1) process with mean μ.

Because the ε's have expectation 0, it follows that $E[Z_n] = 0$.

Is this a Markov process?

The MA(1) process is *not* a Markov process, since

$$Z_n = \varepsilon_n + \theta_1 \varepsilon_{n-1}$$

and

$$\varepsilon_{n-1} = Z_{n-1} - \theta_1 \varepsilon_{n-2}$$

so

$$Z_n = \varepsilon_n + \theta_1 Z_{n-1} - \theta_1^2 \varepsilon_{n-2}.$$

But

$$\varepsilon_{n-2} = Z_{n-2} - \theta_1 \varepsilon_{n-3}$$

so

$$Z_n = \varepsilon_n + \theta_1 Z_{n-1} - \theta_1^2 Z_{n-2} + \theta_1^3 \varepsilon_{n-3}.$$

And continuing, we would see that Z_n depends explicitly on all previous Z's. This means that prediction of Z_n, given Z_{n-1} could be improved upon by making use of earlier Z's, contradicting the Markov property.

Development of the Variance for the MA(1) Process

By squaring both sides of the defining equation and taking expectations, we have

$$E[Z_n^2] = \sigma^2(1 + \theta_1^2).$$

This is the variance of Z_n, because $E[Z_n] = 0$.

Details: Squaring the right hand side gives $\theta_1^2 \varepsilon_{n-1}^2 + \varepsilon_n^2 + 2\theta_1 \varepsilon_n \varepsilon_{n-1}$. Since the ε's are independent, $E[\varepsilon_n \varepsilon_{n-1}] = E[\varepsilon_n]E[\varepsilon_{n-1}] = 0$ so the expected value of the right hand side is $\sigma^2\theta_1^2 + \sigma^2$.

Development of the First Lag Autocovariance

By multiplying the defining equation by Z_{n-1} and taking expectations, show that

$$E[Z_n Z_{n-1}] = \theta_1 \sigma^2.$$

Details: To do this, you will need to use the fact that the defining equation also implies that

$$Z_{n-1} = \theta_1 \varepsilon_{n-2} + \varepsilon_{n-1}.$$

Then

$$Z_n Z_{n-1} = \varepsilon_n \varepsilon_{n-1} + \theta_1 \varepsilon_{n-1}^2 + \theta_1 \varepsilon_n \varepsilon_{n-2} + \theta_1^2 \varepsilon_{n-1} \varepsilon_{n-2}.$$

Taking expectations of this and using independence of the ε's gives

$$E[Z_n Z_{n-1}] = \theta_1 E[\varepsilon_{n-1}^2] = \theta_1 \sigma^2.$$

Thus, the covariance of Z_n and Z_{n-1} is $\theta_1 \sigma^2$. This is the first lag autocovariance.

Since $[Z_n] = 0$, the covariance is just E[$Z_n Z_{n-1}$].

Use the same procedure as in the preceding part to show that

$$E[Z_n Z_{n-k}] = 0$$

for $k = 2, 3, \ldots$. We deduce that the covariance of all autocovariances at lags larger than 1 must be 0.

Details:

$$E[Z_n Z_{n-k}] = E[(\varepsilon_n + \theta_1 \varepsilon_{n-1})(\varepsilon_{n-k} + \theta_1 \varepsilon_{n-k-1})] = 0$$

because all of the products inside the expectation involve different ε's and, therefore, must have expectation 0, because of independence.

The kth autocorrelation is defined as the kth autocovariance divided by the variance of the process. We deduce that all autocorrelations beyond lag 1 are 0, and that the first one is nonzero.

Details: Since the autocovariances beyond lag 1 are all 0, dividing by anything still gives 0. The lag 1 autocovariance is nonzero so dividing by anything still gives a nonzero result.

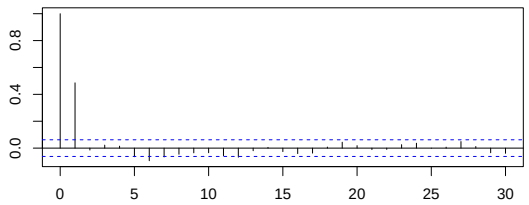
Simulation of MA(1) Data

Simulate 1000 observations from an MA(1) process with $\theta_1 = 0.9$, using the following code

```
ma1 <- arima.sim(1000, model=list(ma=c(.9)))
```

The sample autocorrelation function for the data is obtained as follows:

```
acf(ma1)
```



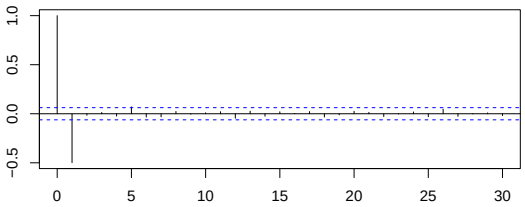
The autocorrelations are all 0 beyond lag 1 as expected, and the first one is positive which agrees with the formula.

Simulating an MA(1) Process with a Negative Parameter

Repeating the preceding simulation but with $\theta_1 = -0.9$:

```
ma1 <- arima.sim(1000, model=list(ma=c(-.9)))
```

```
acf(ma1)
```



The autocorrelations are all 0 beyond lag 1 as expected and the first one is negative which agrees with the formula.

7.3.2 *Distinguishing Between MA(1) and AR(1) Processes*

Given a set of time series data where you are faced with the question as to whether an AR(1) or MA(1) process is appropriate as a model, what action would you take, and how would you make your choice?

Answer: Plot the sample acf function and check to see if the autocorrelations cut off after lag 1 (indicating MA(1)) or if they decay exponentially (indicating AR(1)).

7.3.3 *Fitting an MA Process to Data*

The `arima()` function can be used to fit a moving average model to data.

For the data simulated earlier, in the vector `ma1`, we would use

```
ma1.fit <- arima(ma1, order=c(0, 0, 1))  # the third
# component of order is the MA order
ma1.fit

##
## Call:
## arima(x = ma1, order = c(0, 0, 1))
##
## Coefficients:
##          ma1  intercept
##       -0.8969     0.0021
## s.e.   0.0138     0.0033
##
## sigma^2 estimated as 1.023:  log likelihood = -1431.2,  aic = 2868.4
```

The fitted model is

$$\widehat{Y}_n = \widehat{\mu} + \theta_1 \widehat{\varepsilon}_{n-1}.$$

where ε_{n-1} is the difference (residual) between Y_{n-1} and $\widehat{Y}_{n-1}$. ($\widehat{Y}_0 = \widehat{\mu}$.) Plugging in the results from the output, we have

$$\widehat{Y}_n = -.8946\widehat{\varepsilon}_{n-1}.$$

7.3.4 *Making Predictions*

Predicting the next 5 observations can be done with the predict.ARIMA function as follows:

```
predict(ma1.fit, n.ahead=5)
```

```
## $pred
## Time Series:
## Start = 1001
## End = 1005
## Frequency = 1
## [1] -0.649027767  0.002102616  0.002102616  0.002102616
## [5]  0.002102616
##
## $se
## Time Series:
## Start = 1001
## End = 1005
## Frequency = 1
## [1] 1.011510 1.358764 1.358764 1.358764 1.358764
```

7.3.5 The Moving Average Process of Order 2 (MA(2))

This is defined as

$$Z_n = \theta_2 \varepsilon_{n-2} + \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

where the ε's are independent and normally distributed with mean 0 and variance σ_2, θ_2 and θ_1 are constant.

According to this definition, any ε_k will be independent of Z_j for all $j < k$.

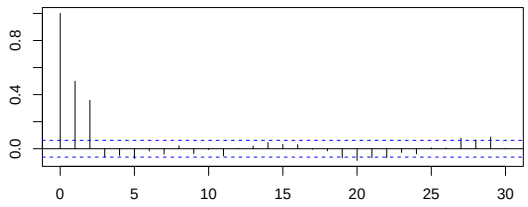
If we define $Y_n = Z_n + \mu$, then Y_n is a MA(2) process with mean μ.

Simulation of MA(2) Data

We can simulate 1000 observations from an MA(2) process with $\theta_1 = 0.5$ and $\theta_2 = 0.7$ using the following code

```
ma2 <- arima.sim(1000, model=list(ma=c(.5, .7)))
```

```
acf(ma2)
```



The autocorrelations are all 0 beyond lag 2 as expected and the first two are nonzero.

The ACF gives us a way to distinguish data following an MA(2) process from an MA(1) process.

Fitting MA(2) Models

Use the `arima()` function to estimate the parameters for given data.

For the `ma2` data, we would use

```
ma2.ma2 <-  arima(ma2, order=c(0, 0, 2)) # Order MA(2)
```

Predicting Future Values in MA(2) Models

We use the `arima.ARIMA()` function to predict the next values in the sequence. To predict the next 10 values of the `ma2` series, we use

```
predict(ma2.ma2, n.ahead=10)
```

```
## $pred
## Time Series:
## Start = 1001
## End = 1010
## Frequency = 1
##  [1] -0.96650802 -1.32365990 -0.06940735 -0.06940735
##  [5] -0.06940735 -0.06940735 -0.06940735 -0.06940735
##  [9] -0.06940735 -0.06940735
##
## $se
## Time Series:
## Start = 1001
## End = 1010
## Frequency = 1
##  [1] 1.005433 1.140352 1.322952 1.322952 1.322952 1.322952
##  [7] 1.322952 1.322952 1.322952 1.322952
```

7.3.6 The autoregressive-moving average process of order 1, 1 (ARMA(1, 1))

This is defined as

$$Z_n = \phi_1 Z_{n-1} + \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

where the ε's are independent and normally distributed with mean 0 and variance σ_2, θ_2 and θ_1 are constant. According to this definition, any ε_k will be independent of Z_j for all $j < k$. If we define $Y_n = Z_n + \mu$, then Y_n is an ARMA(1) process with mean μ.

Simulating ARMA(1,1) Data

Simulate 1000 observations from an ARMA(1, 1) process with $\phi_1 = 0.5$ and $\theta_2 = 0.7$ using the following code

```
arma11 <- arima.sim(1000, model=list(ar=c(.5), ma=c(.7)))
```

Fitting and Predicting with ARMA(1,1) Models

Use the `arima()` function to estimate the parameters. For the `arma11` data set, we would use

```
arma11.arma11 <- arima(arma11, order=c(1, 0, 1))
```

Use the `arima.ARIMA()` function to predict the next values in the series.

7.3.7 The Integrated Autoregressive-Moving Average Process (ARIMA)

ARIMA processes allow for random trends. The ARIMA(1, 1, 1) is defined as

$$X_n = X_{n-1} + Z_n$$

where

$$Z_n = \phi_1 Z_{n-1} + \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

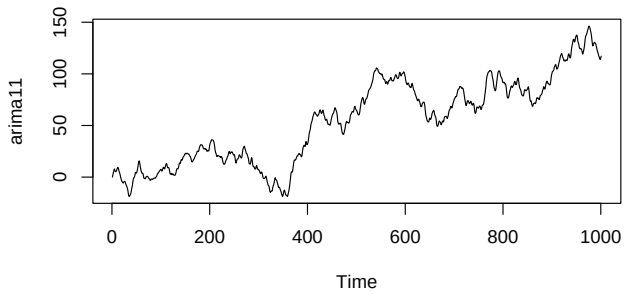
and where the ε's are independent and normally distributed with mean 0 and variance σ_2, θ_2 and θ_1 are constant. An ARIMA(1,1,1) process with drift μ is defined as $X_n = X_{n-1} + Y_n$ where $Y_n = Z_n + \mu$; this incorporates a deterministic trend as well as a random trend.

Simulating ARIMA Data

We can simulate 1000 observations from an ARMA(1, 1, 1) process with $\phi_1 = 0.5$ and $\theta_2 = 0.7$ using the following code

```
arima11 <- arima.sim(1000, model=list(order=c(1, 1, 1),
     ar=c(.5), ma=c(.7)))
```

```
ts.plot(arima11)
```



Note the apparent trend. This is not systematic and could just as likely trend downward.

Fitting and Predicting with ARIMA Models

Use the `arima()` function to estimate the parameters. For the simulated data in `arima11`, we would use

```
arima11.arima <- arima(arima11, order=c(1,1,1))
# the middle one indicates 1 integration order
```

Use the `arima.ARIMA()` function to predict the next values in the series.

Changing the Noise Standard Deviation

In order to simulate data with a different noise standard deviation, use the `sd` argument in the `arima.sim()` function as, for example, with $\sigma = 10$:

```
arima11 <- arima.sim(1000, model=list(order=c(1, 1, 1),
    ar=c(.5), ma=c(.7)), sd=10)
```

Automatic Fitting of ARIMA Models Using AIC

The `auto.arima()` function in the *forecast* package uses AIC (and related criteria) to automatically choose from among the different models. Models with smaller AIC values are preferred. The AIC criterion balances the goodness of fit of the model to the data via maximum likelihood estimation with a penalty on the complexity of the model. In other words, a model with many parameters, that is, a very complex model, might fit the data very well, while a model with only a few parameters is simple but might not fit the data well. AIC strikes a balance between these simplicity and goodness of fit.

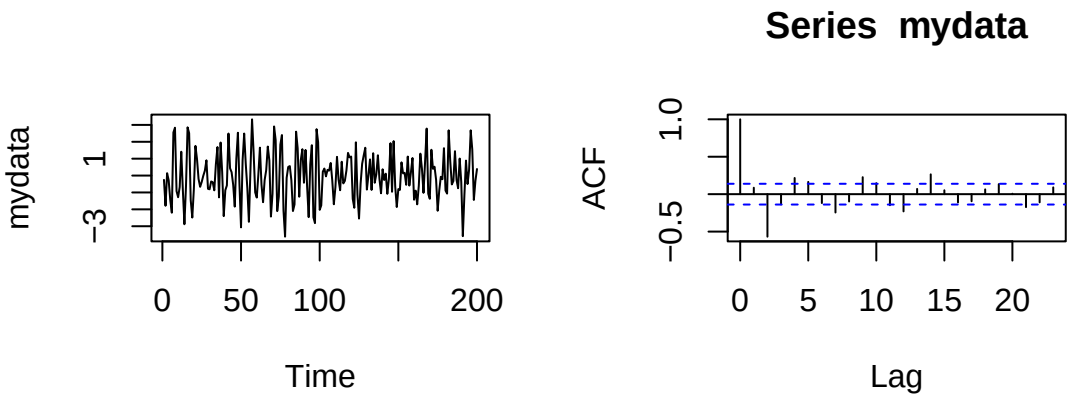
7.3.8 Some Illustrative Examples

```
mydata <- arima.sim(200, model=list(ma=c(.1, -.85))) # MA(2) data
library(forecast)
auto.arima(mydata)

## Series: mydata
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##          ar1      ar2
##       0.1363  -0.5785
```

```
## s.e.  0.0575   0.0572
##
## sigma^2 estimated as 1.349:  log likelihood=-313.12
## AIC=632.24   AICc=632.37   BIC=642.14
```

Plotting Simulated MA(2) Data

```
par(mfrow=c(1,2))
ts.plot(mydata); acf(mydata)
```

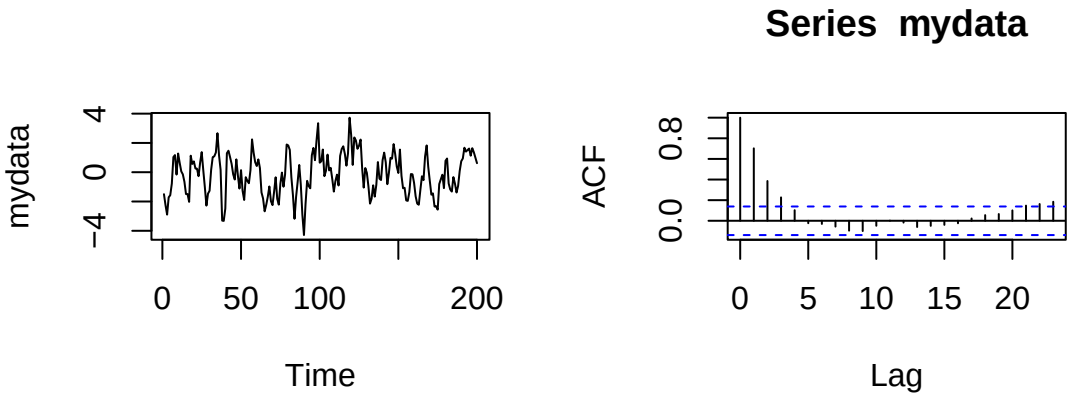


Simulating and Fitting ARMA(1,1) Data

```
mydata <- arima.sim(200, model=list(ar=c(.6), ma=c(.3))) # ARMA(1,1)
auto.arima(mydata)

## Series: mydata
## ARIMA(1,0,1) with zero mean
##
## Coefficients:
##          ar1     ma1
##       0.5443  0.3416
## s.e.  0.0780  0.0863
##
## sigma^2 estimated as 0.9672:  log likelihood=-279.86
## AIC=565.72   AICc=565.84   BIC=575.62
```

Plotting Simulated ARMA(1,1) Data

```
par(mfrow=c(1,2))
ts.plot(mydata); acf(mydata)
```

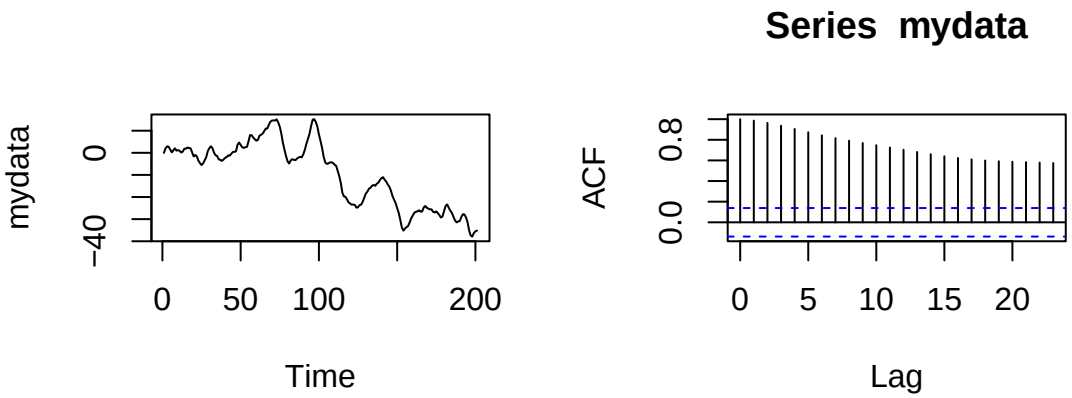


Simulating and Fitting ARIMA(1,1,1) Data

```
mydata <- arima.sim(200, model=list(order=c(1,1,1), ar=c(.6), ma=c(.3))) # ARIMA(1,1) data
auto.arima(mydata)

## Series: mydata
## ARIMA(1,1,1)
##
## Coefficients:
##          ar1     ma1
##       0.5858  0.3513
## s.e.  0.0763  0.0930
##
## sigma^2 estimated as 0.9903:  log likelihood=-282.27
## AIC=570.54   AICc=570.67   BIC=580.44
```

Plotting Simulated ARIMA(1,1,1) Data

```
par(mfrow=c(1,2))
ts.plot(mydata); acf(mydata)
```

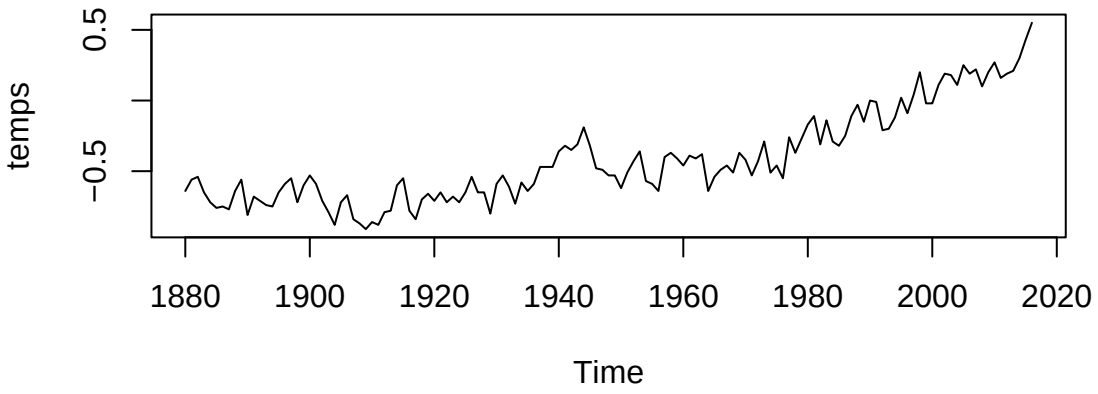


The ACF has no meaning here, since this process is not stationary - it has random trends as seen on the plot on the left.

7.3.9 Application to Global Warming

The data in `Globaltemps.R` are the differences in the global average temperature from that of 1990 for the years 1880 through 2016.

```
source("Globaltemps.R")
temps <- ts(temps, start = 1880, end = 2016)
ts.plot(temps)
```



Data are from Datahub (`https://datahub.io/core/global-temp`).

7.3.10 Fitting an ARIMA(1,1,1) Model

```
temps.arima <- arima(temps, order = c(1,1,1))
temps.arima

##
## Call:
## arima(x = temps, order = c(1, 1, 1))
##
## Coefficients:
##          ar1      ma1
##       0.3518  -0.7017
## s.e.  0.1443   0.1044
##
## sigma^2 estimated as 0.01036:  log likelihood = 117.67,  aic = -229.34
```

This assures us that there is nonstationarity in the data - of the kind that has probably always been operating for millenia - i.e. random trends.

Is there a Deterministic Trend?

A deterministic trend in a nonstationary model is called `drift`.

We can informally check to see if there are trends with slope .001, .003, .005, .007 or .009 in the 137 observations by subtracting such trends out and comparing the resulting AIC values. (Remember we want to minimize AIC.)

```
temps.arima <- arima(I(temps-.001*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -230.2671

temps.arima <- arima(I(temps-.003*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -231.9157

temps.arima <- arima(I(temps-.005*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -233.1589

temps.arima <- arima(I(temps-.007*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -233.7991

temps.arima <- arima(I(temps-.009*(1:137)), order = c(1,1,1))
temps.arima$aic

## [1] -233.7044
```

It looks like there might be drift value of about .007. (You could also use `auto.arima()` to get a similar result.)

Using Simulation to Compare Scenarios

Our fitted model output:

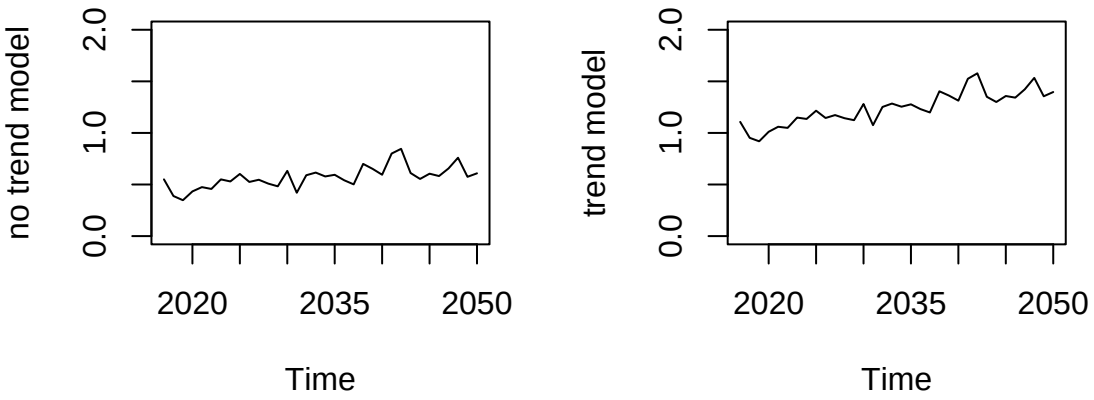
```
temps.arima <- arima(I(temps-.007*(1:137)), order = c(1,1,1))
temps.arima

##
## Call:
## arima(x = I(temps - 0.007 * (1:137)), order = c(1, 1, 1))
##
## Coefficients:
##          ar1      ma1
##       0.3937  -0.7752
## s.e.  0.1314   0.0880
##
## sigma^2 estimated as 0.01001:  log likelihood = 119.9,  aic = -233.8
```

Comparing No Trend Scenario at 2050 with Trend Scenario

```
n <- 34 # forecast to 2050 from 2016
temp.sim <- arima.sim(n-1, model=list(order=c(1,1,1),
    ar=c(.394), ma = -.775), sd = sqrt(.01)) + temps[137]
temptrend.sim <- temp.sim +.007*(1:n) + temps[137]
notrendsim <- ts(temp.sim, start=2017,
     end=2017+n-1) # convert to ts object
trendsim <- ts(temptrend.sim, start=2017,
     end=2017 + n-1) # convert to ts
```

Single Simulation Realizations from the Two Possible Scenarios

```
par(mfrow=c(1,2))
ts.plot(notrendsim, ylim=c(0,2), ylab="no trend model")
ts.plot(trendsim, ylim=c(0,2), ylab="trend model")
```



Projected Temperature Increase by 2050

Repeated simulations under the two scenarios allows us to make a comparison at any percentile level we wish, such 2.5%, 50% and 97.5%, as here:

```
Nsims <- 10000
temp2050 <- numeric(Nsims)
for (j in 1:Nsims) {
    temp.sim <- arima.sim(n-1, model=list(order=c(1, 1,1),
         ar=c(.394), ma = -.775), sd = sqrt(.01)) + temps[137]
    temp2050[j] <- temp.sim[n]
}
quantile(temp2050, c(.025, .5, .975)) # no trend

##       2.5%        50%      97.5%
## 0.06428848 0.54777866 1.04056258

quantile(temp2050+.007*n, c(.025, .5, .975))  # with trend

##      2.5%       50%     97.5%
## 0.3022885 0.7857787 1.2785626
```

These are empirical projections and do not incorporate any of the science behind global circulation models.

Projected Temperature Increases by 2100

The same kind of simulation exercise can be carried out for projections to 2100:

```
temp2100 <- numeric(Nsims)
n <- 84 # number of years from 2016 to 2100
for (j in 1:Nsims) {
    temp.sim <- arima.sim(n-1, model=list(order=c(1,1,1),
```

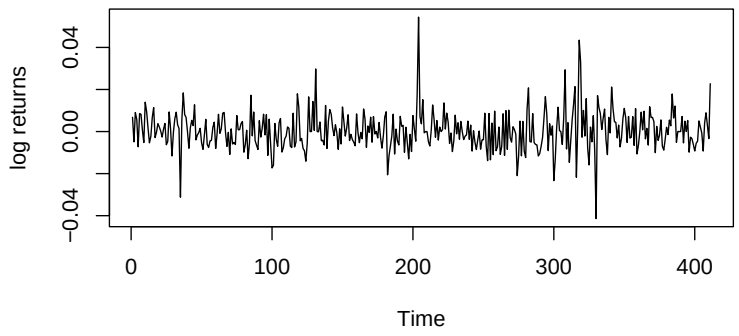
```
          ar=.394, ma = -.775), sd = sqrt(.01)) + temps[137]
    temp2100[j] <- temp.sim[n]
}
quantile(temp2100, c(.025, .5, .975)) # no trend

##       2.5%        50%      97.5%
## -0.1671178  0.5447117  1.2452512

quantile(temp2100+.007*n, c(.025, .5, .975))  # with trend

##      2.5%       50%     97.5%
## 0.4208822 1.1327117 1.8332512
```

7.4 Another Markov Process - ARCH model

The ARCH model, and its more sophisticated variant, GARCH, are important models used to analyze financial time series, such as stock indices and treasury bond yields.

The following trace plot shows daily log returns for the FTSE (Financial Times Stock Exchange) for 1991 and 1992:

```
logreturns <- diff(log(EuStockMarkets[1:412, 4]))
ts.plot(logreturns, ylab="log returns")
```



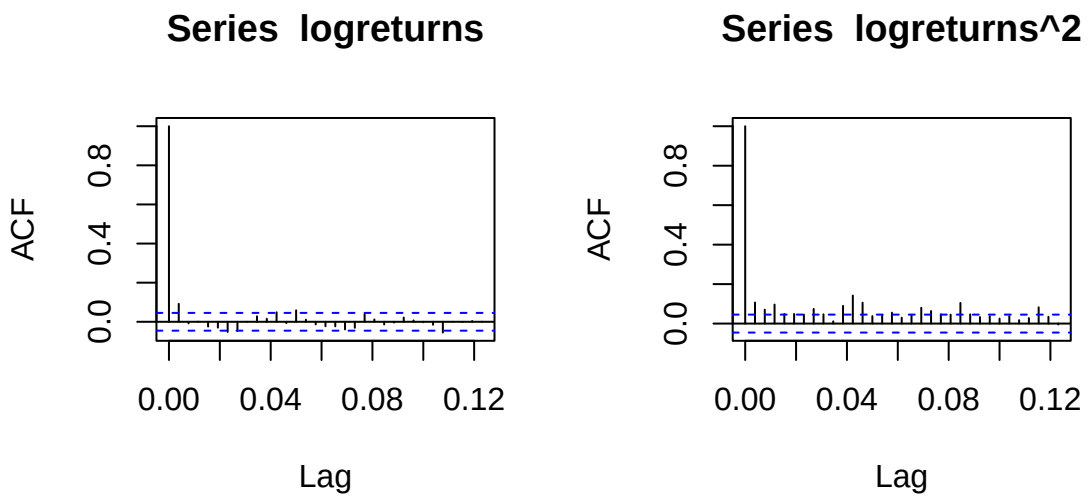
The log return for day t is the logarithm of x_t/x_{t-1}. It gives an indication as to how well the market is doing. A positive log return means that the market went up.

7.4.1 ACF of the Log Returns

Analyze the full data set:

```
logreturns <- diff(log(EuStockMarkets[, 4]))
```

```
par(mfrow=c(1,2))
acf(logreturns)
acf(logreturns^2)
```



There is a slight autocorrelation at lag 1 in the raw log returns, but there is somewhat more autocorrelation in the squared log returns (right panel).

7.4.2 The ARCH model

A model which gives similar behaviour to the daily log returns for the FTSE is the following:

For day t, the log return is given by

$$y_t = s_t Z_t$$

where

$$s_t = \sqrt{a_0 + a_1 y_{t-1}^2 + a_2 y_{t-2}^2 + a_3 y_{t-3}^2}$$

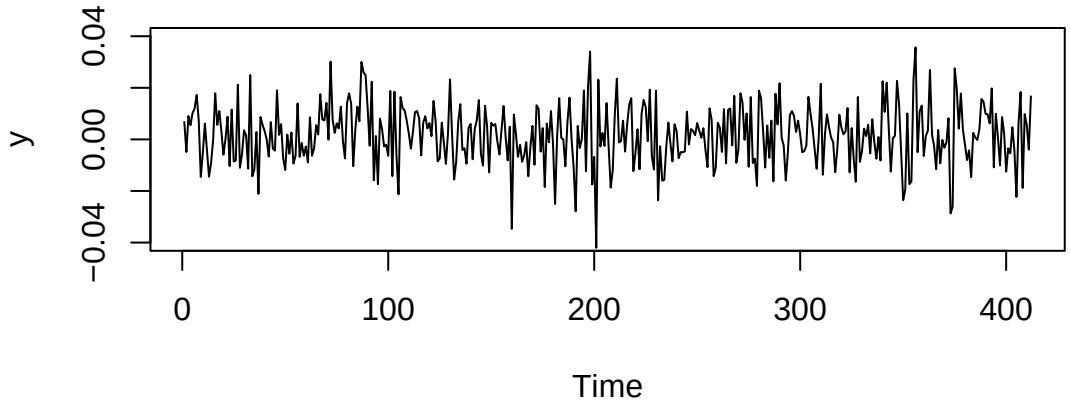
and Z_t is a standard normal random variable.

The parameter values are $a_0 = 0.00001$, $a_1 = 0.1$ and $a_2 = 0.05$ and $a_3 = .08$. Note this model is a lot like an autoregressive process of order 3 in terms of y_t^2.

We can start of a simulation by setting the first three values of y_t to the first three values from the observed log returns.

```
n <- 412 # number of days for the simulation
a <- c(.1, .05, .08); a0 <- .0001
y <- numeric(n); y[1:3] <- logreturns[1:3]
for (t in 4:n){
    s <- sqrt(a0 + a[1]*y[t-1]^2 + a[2]*y[t-2]^2 +
        a[3]*y[t-3]^2)
    y[t] <- s*rnorm(1)
}
```

```
ts.plot(y, ylim=c(-.04, .04))
```



As in the actual series, the simulated values hover between $\pm.01$ but occasionally between $\pm.04$. We might use this model to predict future behaviour of the FTSE, such as how long it might take to exceed some value, such as .04. Suppose we want to simulate the ARCH process until the first time it exceeds 0.04. We don't know beforehand when this will occur, we wouldn't know how to stop the `for()` loop at the right time, so we should use a `while()` loop.

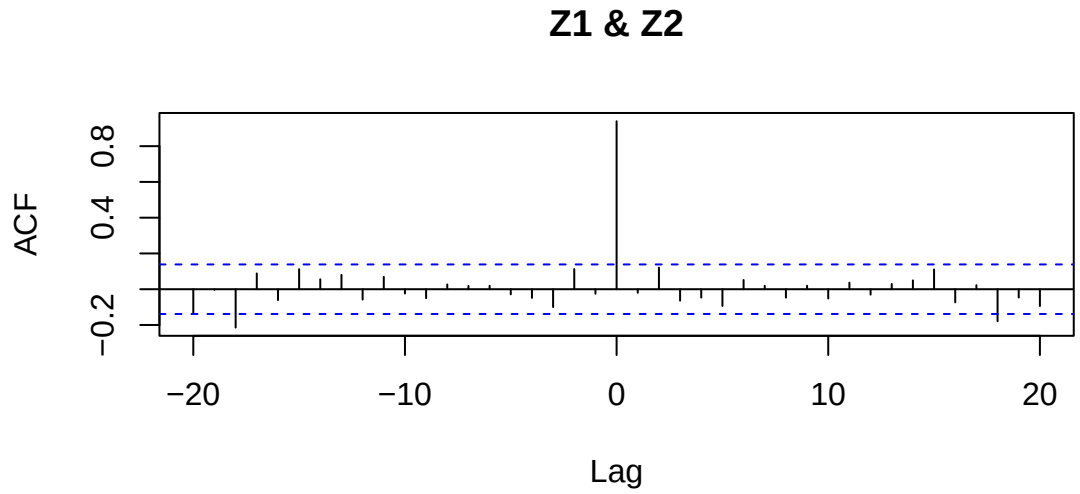
```
a <- c(.1, .05, .08); a0 <- .0001
y <- numeric(n); y[1:3] <- logreturns[1:3]
while (y[t] < .04){
    t <- t+1
    s <- sqrt(a0 + a[1]*y[t-1]^2 + a[2]*y[t-2]^2
        +a[3]*y[t-3]^2)
    y[t] <- s*rnorm(1)
}
print(t)

## [1] 1326
```

By repeatedly running this simulation, we could obtain a distribution of the times until we would expect the log returns to first exceed .04. This kind of information would be useful, for example, in pricing certain options.

7.5 Cross-Correlation

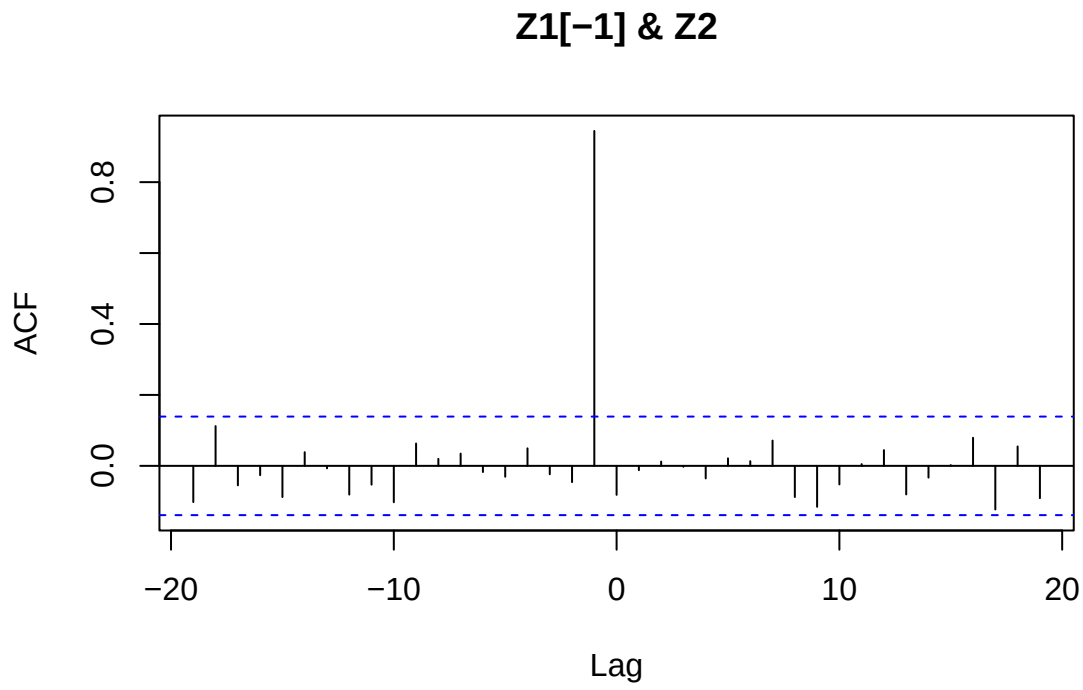
e.g. 2 series, correlated at the same lag (lag 0)

```
Z1 <- rnorm(200)
Z2 <- 3*Z1 + rnorm(200)
ccf(Z1, Z2)
```



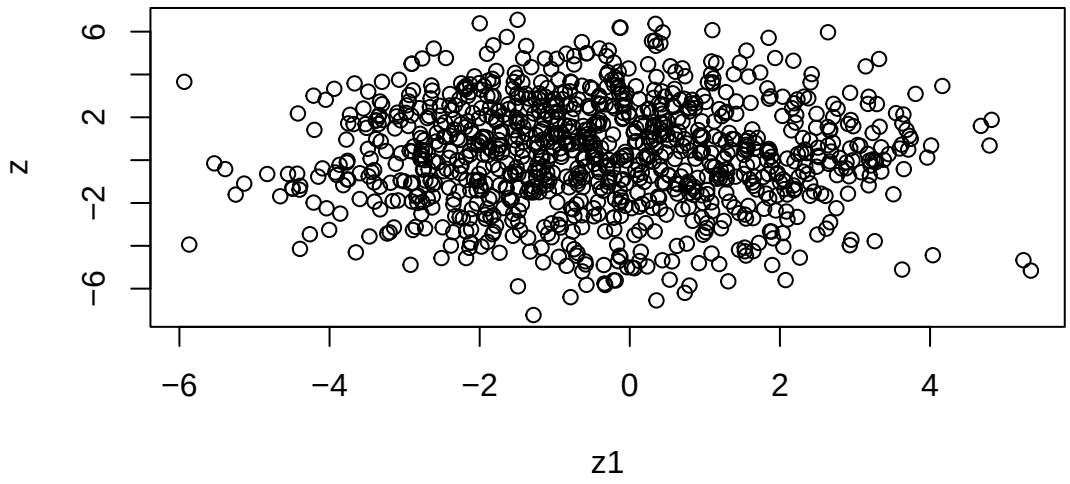
e.g. 2 series, correlated at lag 1

```
Z1 <- rnorm(200)
Z2 <- 3*Z1[-200] + rnorm(199)
ccf(Z1[-1], Z2)
```



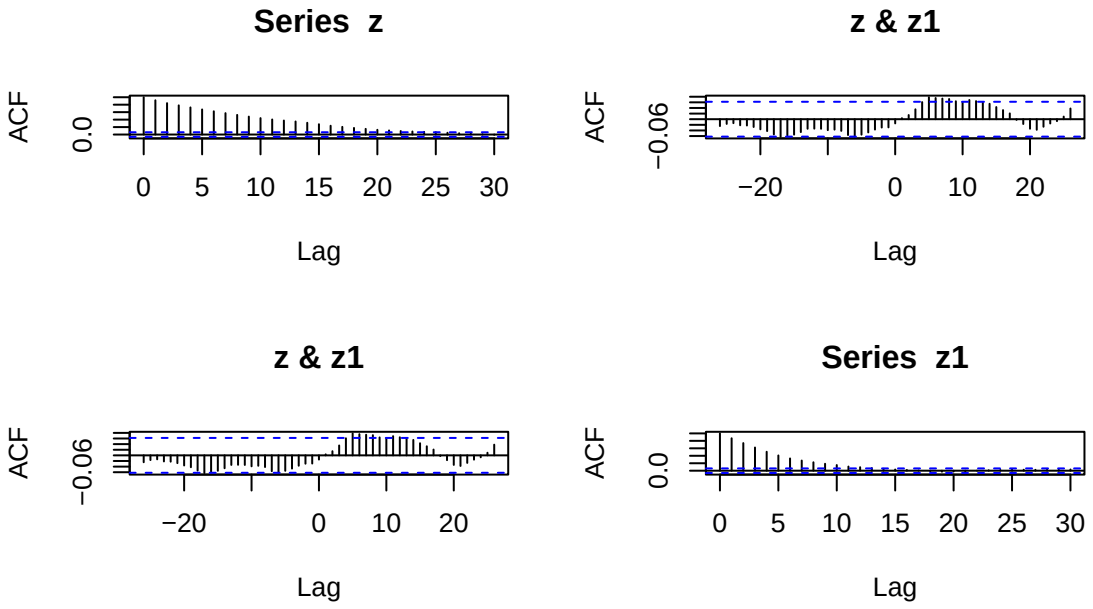
Cross-correlations can easily be misinterpreted if the series are not pre-whitened.

```
z <- arima.sim(1000, model=list(ar=c(.9)))
z1 <- arima.sim(1000, model=list(ar=c(.9))) # z and z1 are independent time series
plot(z ~ z1);  cor(z, z1)
```




```
## [1] -0.01498715
```

```
par(mfrow=c(2,2))
acf(z);  ccf(z, z1); ccf(z, z1); acf(z1)
```



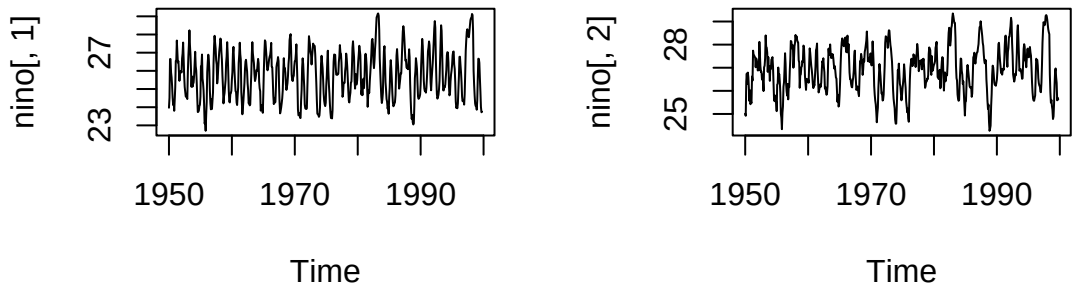
7.5.1 Nino Data Set

Monthly temperature data for two regions of the Pacific Ocean

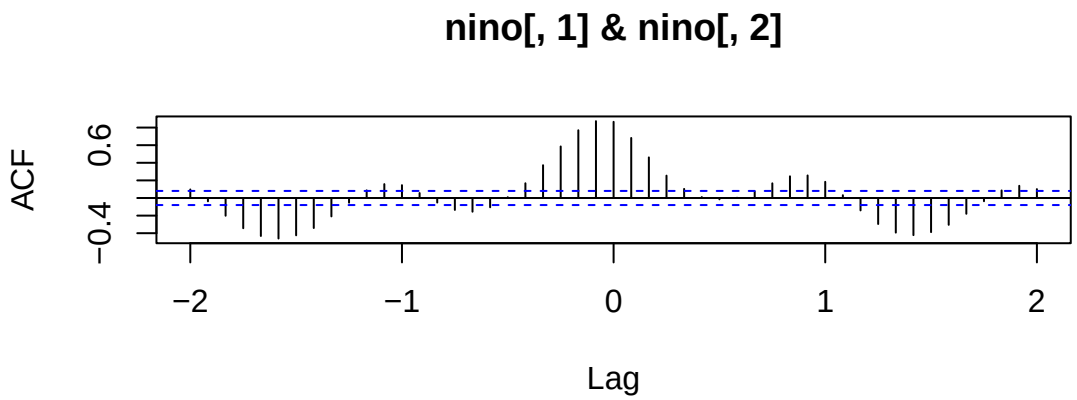
```
library(tseries)  # load time series package
data(nino) # load nino data set
summary(nino)

##      nino3           nino3.4
##  Min.   :22.70   Min.   :24.27
##  1st Qu.:24.72   1st Qu.:26.28
##  Median :25.80   Median :26.95
##  Mean   :25.77   Mean   :26.94
##  3rd Qu.:26.70   3rd Qu.:27.58
##  Max.   :29.16   Max.   :29.35
```

The Nino 3 Region is bounded by 90W-150W and 5S-5N. The Nino 3.4 Region is bounded by 120W-170W and 5S-5N.

```
par(mfrow=c(1, 2))
ts.plot(nino[,1])
ts.plot(nino[,2])
```



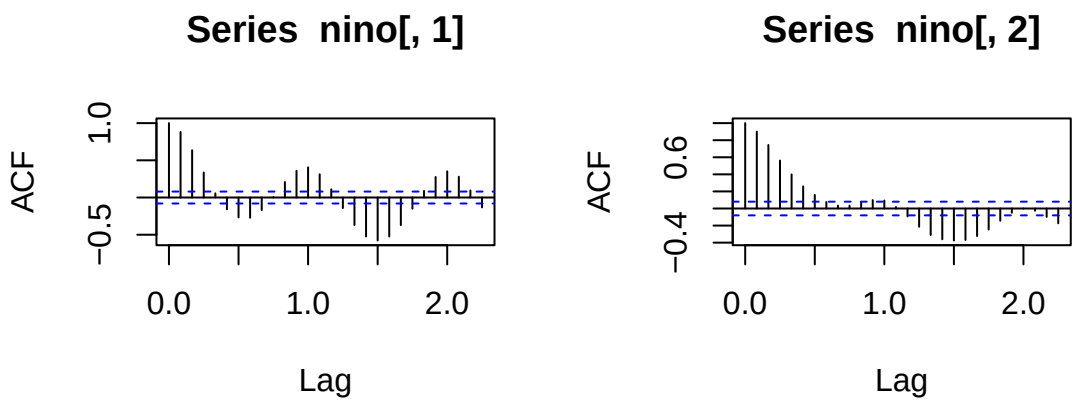

```
ccf(nino[,1], nino[,2])
```



According to this, it looks like you might be able to predict in one region from the other at a lag of about 1.5 years. That is, summer in one region is connected to winter in the other region. Seems weird. Is this real?

Autocorrelations

```
par(mfrow=c(1, 2))
acf(nino[,1])
acf(nino[,2])
```



The two series have internal dependencies which should be modelled before relating them to each other.

Modeling the two series

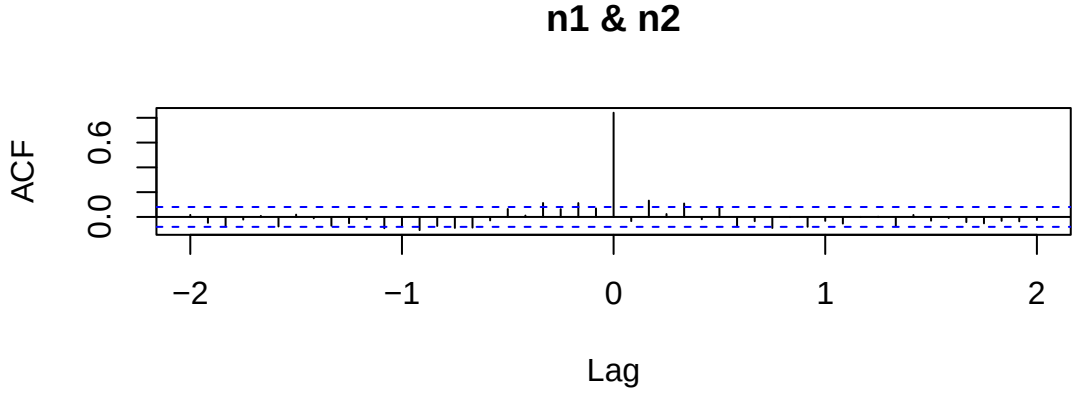
Because there is a seasonal effect, we incorporate this on top of an AR(2) monthly model. We find the residuals directly. This is *pre-whitening*.

```
n1 <- resid(arima(nino[,1], order=c(2, 0, 0),
    seasonal=list(order=c(1, 0, 1))))
n2 <- resid(arima(nino[,2], order=c(2, 0, 0),
    seasonal=list(order=c(1, 0, 1))))
```

How are the two series related?

Cross-correlation between residuals from the two series:

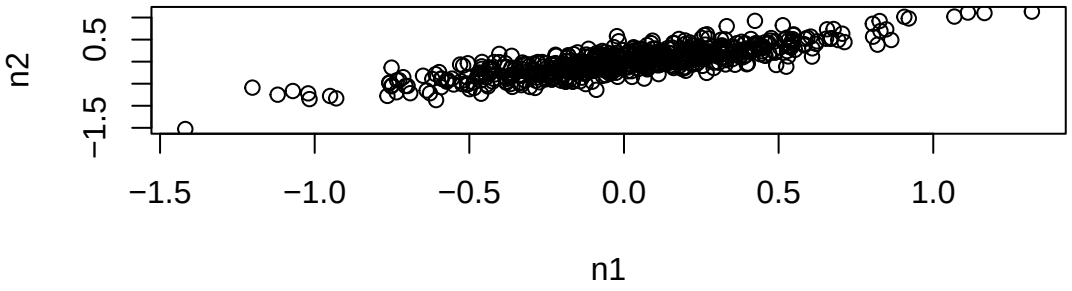
```
ccf(n1, n2)
```



They are related, but really only at lag 0. In other words, current information in one region could be used to predict behaviour in the other region. But we no longer see predictive possibilities using older information.

Here is a plot of the residuals after pre-whitening.

```
plot(n1, n2)
```



The one set of residuals nicely predicts the other set.

Exercises

1. Using a `for()` loop and the `arima.sim()` function, simulate 1000 time series (of length 100) which mimic the Lake Huron levels. ... write this example so that we seek the probability of a lake level following below 575 feet and compare with the use of the independence model.
2. Suppose Z_n satisfies the conditions of an MA(2) model as follows:

$$Z_n = \theta_1 \varepsilon_{n-1} + \theta_2 \varepsilon_{n-2} + \varepsilon_n$$

   where the ε's are independent with mean 0 and variance σ^2.

   (a) Find the expected value of Z_n, and its variance.
   (b) Find the autocovariances at lags 1, 2 and 3.
   (c) Conclude that the lag 3 autocorrelation is 0.
3. Suppose Z_n satisfies the conditions of a stationary ARMA(1,1) model as follows:

$$Z_n = \phi_1 Z_{n-1} + \theta_1 \varepsilon_{n-1} + \varepsilon_n$$

   where the ε's are independent with mean 0 and variance σ^2. We assume $\phi \in (-1, 1)$ here.

   (a) Find the expected value of Z_n.
   (b) Show that $E[Z_n \varepsilon_n] = \sigma^2$.
   (c) Find the autocovariances at lags 1, 2 and 3.
   (d) Guess a recursion that could be used to compute the kth lag autocovariance for the ARMA(1,1) process, from the $(k-1)$st lag autocovariance, for $k > 1$.
   (e) What does this tell you about the autocorrelations for an ARMA(1, 1) process as k increases?
4. The file `ffmc.R` contains a list of three lists, each containing 36 time series of measurements of the file fuel moisture code at 3 weather stations in northwestern Ontario, an area prone to large wildfires.

   (a) Load the data into R as follows:

```
source("ffmc.R")
```

   Then consider the 1963 time series for station 10400, which is in the `year1963` list within the list `ffmc400`. Assign this time series to the object `FFMC`.

(b) Construct a trace plot of the data. Is there any obvious trend?

(c) Construct a plot of the autocorrelation function. Is there evidence of linear dependence within the data?

(d) Use `auto.arima()` in the *forecast* package to select a model for these data. Write down the fitted model.

(e) Simulate data from the fitted model, compare the trace plot and acf with the corresponding plots for the original data. Do you think there is an issue with your fitted model?

(f) In fact, the fine fuel moisture code is inversely related to percent fuel moisture m, according to the relation

$$m = 101 - \text{FFMC}.$$

Create an object called `m` and construct a trace plot.
Then take the log and construct a trace plot of the result.
Why do you think $\log(m)$ than will be easier to analyze than m or FFMC using an ARIMA model?

(g) Use `auto.arima()` to select a model for $\log(m)$. Hence, or otherwise, fit an ARMA(1,1) model to $\log(m)$.

(h) Simulate from the fitted model, and transform the simulated data back to the form of simulated FFMC, and compare the trace plot and acf plot with the plots for the original data. Do you see an improved fit to the data?

5. Using the fitted model from the previous question, simulate 1000 time series having the same length as the original 1963 ffmc series, remembering to back-transform so that the resulting series simulate the FFMC values. For each simulated series, use the `sum()` function (or alternative methods if you prefer) to calculate the number, N, of days for which the simulated FFMC values exceed 90. Estimate the expected value of N and standard deviation of N. Compare the standard deviation of the simulated value with what would be obtained if N is assumed to be a binomial random variable, i.e. using $\sqrt{np(1-p)}$, where n is the length of the 1963 series and p is an estimate of the proportion of days where the simulated FFMC exceeds 90.

   In 1973, there were 13 days where the FFMC exceeded 90. Based on the analysis that you have done in this question, would you conclude that this is an unusual event, or is it more-or-less expected?

6. Use `arima()` to fit ARMA(1,1) models to the other time series in the `ffmc$ffmc400` list. Plot the time series coefficients against year. Do these fits give you more, or less, confidence in the appropriateness of your 1963 model? (It is possible to do this with a for loop.)

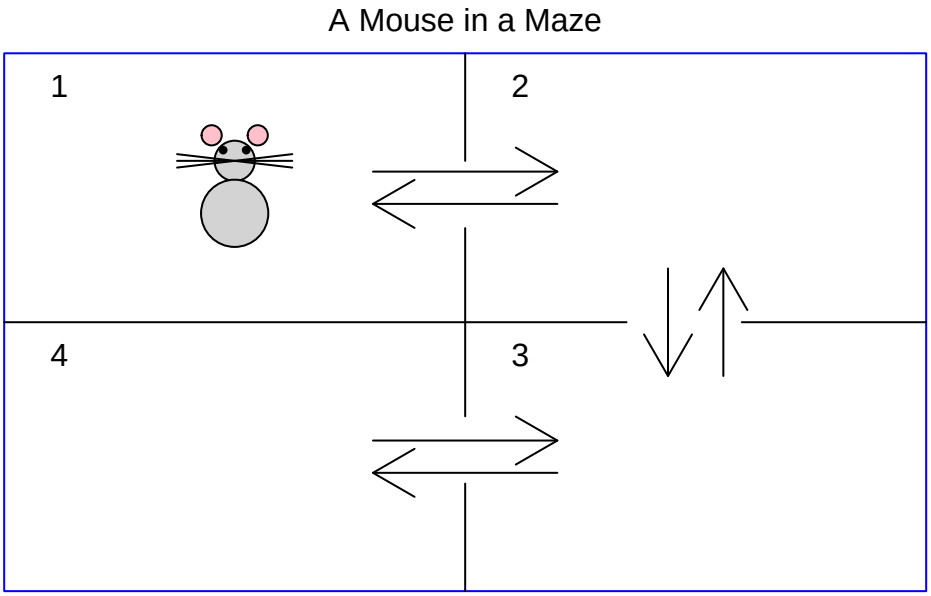
7. Repeat the analysis of the previous exercise on the data in `ffmc600` and `ffmc602`.

8

Discrete Time Markov Chains

8.1 An Illustrative Example

We begin with a very simple example that illustrates many of the concepts that arise when studying and simulating from models. The figure below represents the floor plan for a simple maze, consisting of four adjoining compartments, in which a mouse may randomly wander around in. 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 between compartments 2 and 3, and between compartments 3 and 4.



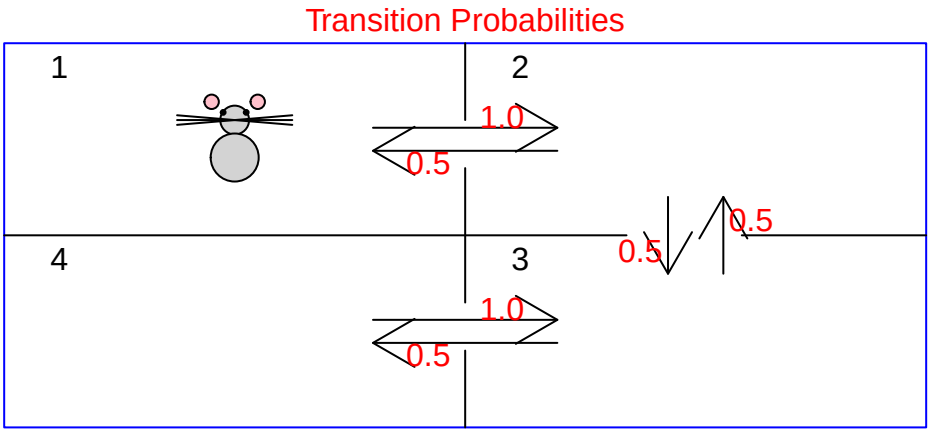
8.1.1 A Mouse Movement Model - Complete Randomness

If the mouse wanders aimlessly, we might attach probabilities to the transitions between compartments in the maze as in the following diagram. Alternatively, the mouse might favour one of the compartments over another, perhaps because of the location of food or a particular odor.

In the figure below, we assume complete randomness. In other words, the mouse is just moving randomly around, stopped only by the walls between the compartments. 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. 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 the figure below.



8.1.2 *Transition Matrix*

The transition probabilities can be organized systematically into an array, called the , noting that impossible transitions have probability 0. The 1st 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.

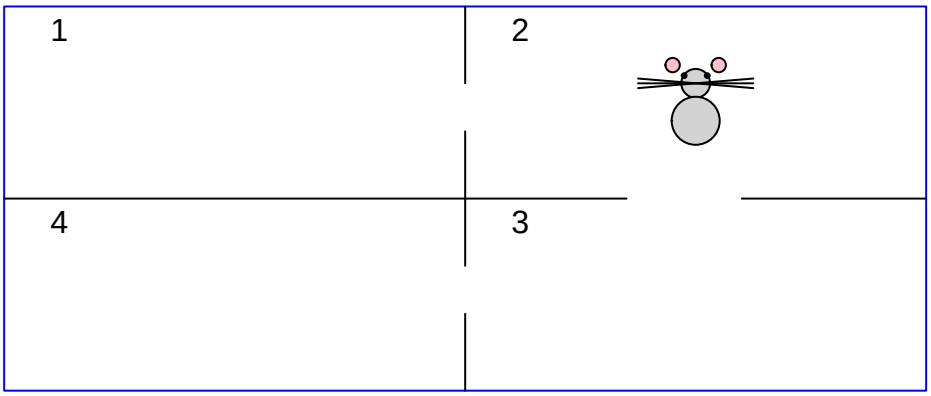
The transition matrix for this model is then

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

Entry (i, j) represents the probability of transition into compartment j, given that the mouse was in state i.

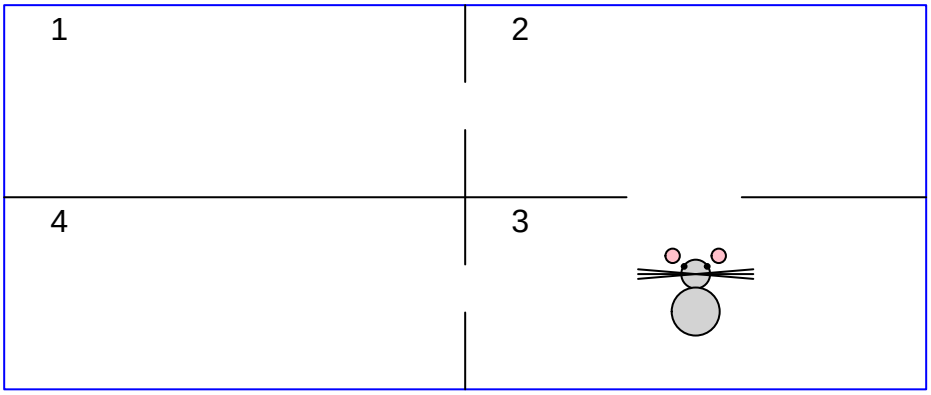
8.1.3 *Simulating from the Mouse Movement Model*

For any time $t = 0, 1, 2, \ldots$, we define X_t as the compartment holding the mouse at time t. $X_0 = 1$, since the mouse starts in the first compartment. At the first transition, the mouse enters the second compartment: $X_1 = 2$ as seen in the figure below.



At the second transition, the mouse can enter compartment 1 with probability 0.5 or compartment 3 with probability 0.5, like a Bernoulli trial. To simulate the transition, we need to generate a pseudorandom number which we will call U_2. If $U_2 < 0.5$, the mouse enters compartment 1. Otherwise, it enters compartment 3.

Supposing we obtain the value $U_2 = 0.61$, then the mouse enters the 3rd compartment: $X_2 = 3$.



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

The next three simulated values are .34, .88 and .52. The outcomes for the Markov chain are:

- $U_3 = 0.34$, so the mouse enters the 2nd compartment: $X_3 = 2$.
- $U_4 = 0.88$, so $X_4 = 3$.
- $U_5 = 0.52$, so $X_5 = 4$.

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.

8.1.4 Simulating from the model in R

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.

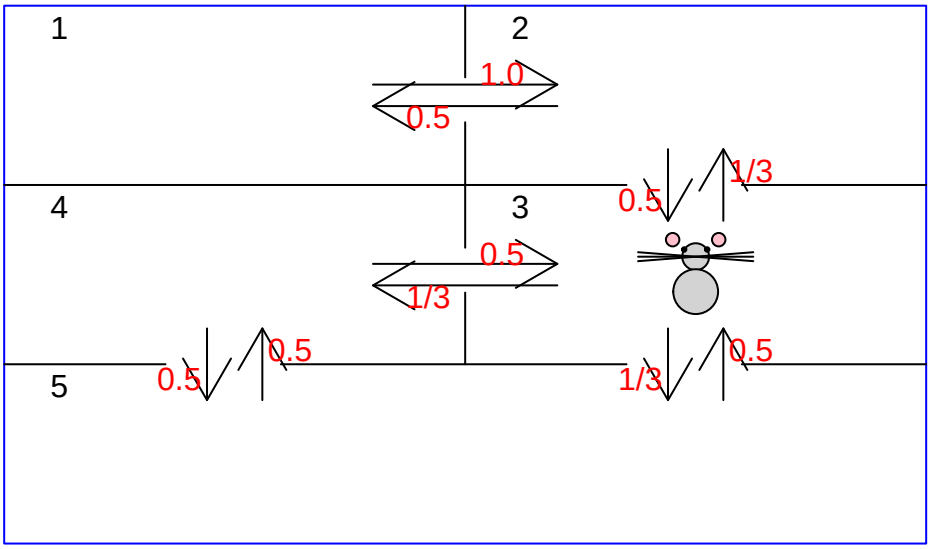
Note that when simulating values in a Markov chain, we are simulating from probability distributions defined by the rows of the transition matrix.

```
Ntransitions <- 100000 # number of mouse moves
P <- matrix(c(0, 1, 0, 0,
              0.5, 0, 0.5, 0,
              0, 0.5, 0, 0.5,
              0, 0, 1, 0), nrow = 4,
              byrow = TRUE)  # P is the transition matrix
location <- numeric(Ntransitions)#initializing the chain
current.state <- 1 # initial compartment
for (t in 1:Ntransitions) {
   current.state <- sample(1:4, size = 1, prob = P[current.state, ])
   location[t] <- current.state
}
```

```
table(location) # this counts visits to each compartment

## location
##     1     2     3     4
## 16578 33151 33422 16849
```

8.1.5 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.

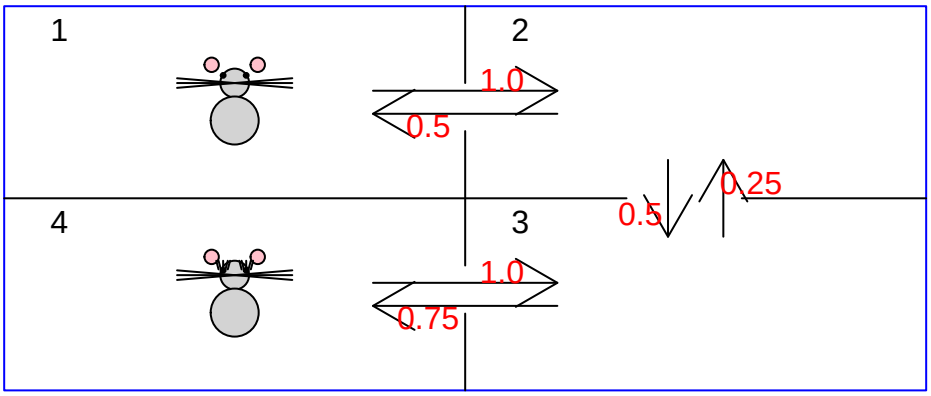
```
P5 <- matrix(c(0, 1, 0, 0, 0,   0.5, 0, 0.5, 0, 0,
               0, 1/3, 0, 1/3, 1/3,   0, 0, 0.5, 0, 0.5,
               0, 0, 0.5, 0.5, 0), nrow = 5, byrow = TRUE)
location <- numeric(Ntransitions); current.state <- 1
for (t in 1:Ntransitions) {
       current.state <- sample(1:5, size = 1, prob = P5[current.state, ])
       location[t] <- current.state
}
```

```
table(location) # numbers of visits to each of the five compartments

## location
##     1     2     3     4     5
## 10071 20161 30068 19839 19861
```

8.1.6 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
## 10074 20106 39926 29894

    # 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.

8.1.7 *Long run distribution*

We have seen that when we repeatedly simulate these Markov chains, we will find that the proportions of visits to each location follow specific distributions. In the null model with four compartments, this *steady state* distribution is $1/6, 1/3, 1/3, 1/6$. In the five compartment model and when there is an odor, the steady state distribution is different. (We will find out how to calculate it later.)

8.2 Definitions and terminology

Definition. *State Space* = $S = \{1, 2, \ldots, m\}$.

Example 8.1 *For the mouse example,* $S = \{1, 2, 3, 4\}$.

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.

Example 8.2 *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.*

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

Define a matrix P with (i, j)th entry

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

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

All rows of P sum to one. This is equivalent to noting that

$$P \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Example 8.3 *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)
```

8.3 Probability calculations

The (conditional) probability distribution of the nth variable in the Markov chain has a simple interpretation in terms of the transition matrix: $P(X_n = j|X_0 = i)$ is the (i, j)th element of the matrix P^n.

Example 8.4
```
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.

Example 8.5 *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,*
$x^{\{0\}} = [.25, .25, .25, .25]$.

The (unconditional) probability distribution of the Markov chain at the nth time step can be calculated using

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

Example 8.6 $x^{\{0\}} P_{33}$ *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
```

$x^{\{0\}}$, $x^{\{1\}}$ *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
```

Probability distribution of X_n *when* n *is large*

We can calculate the probability distribution of X_n for fairly large n by taking powers of the transition matrix often enough:

```
P2 <- P33%*%P33
P4 <- P2%*%P2 # 4th power of P
P8 <- P4%*%P4 # 8th power of P
P16 <- P8%*%P8 # 16th power of P
x16 <- x0%*%P16 # distribution after 16 transitions
x16

##           [,1]       [,2]      [,3]
## [1,] 0.2173913 0.08695657 0.6956522

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 long run distribution or steady state distribution of this Markov chain.

8.3.1 Does every Markov chain have a long run distribution?

Not all Markov chains have long run 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 8.7
```
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 8.8 *P for the mouse odor example is not regular. Therefore, regularity is not always necessary for a long run distribution to exist.*

Example 8.9 *P_5 for the five-compartment mouse maze model is regular. Therefore, a long run distribution exists.*

Implications of regularity

If P is a regular matrix, then there exists a vector $\mathbf{q}$ such that

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

Example 8.10
```
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 8.11 `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 $\mathbf{q}$ such that

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

for any initial state vector $x^{\{0\}}$. The vector $\mathbf{q}$ specifies the long run distribution of the Markov chain. If P is a regular matrix, then the long run distribution vector $\mathbf{q}$ 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 *steady state* vector.

Example 8.12
```
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 8.13 *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
```

so the steady state distribution still satisfies

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

8.3.2 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 $\mathbf{q}$. (One implication of this is Markov Chain Monte Carlo simulation, i.e. MCMC).

8.3.3 Periodic Markov chains

The four-compartment mouse maze model is an example of a periodic Markov chain. Starting in compartment 1 at time 0, it is impossible for the mouse to return to compartment 1 at any odd-valued time. Similarly, it is impossible for the mouse to return to compartment 2 at any even-valued time, and this is also true of compartment 4. Compartment 3 can only be returned to at even-valued times. Therefore, this model is an example of a periodic Markov chain with period 2.

Markov chains periods of any positive integer value, including 1. Markov chains with period 1 are referred to as aperiodic. Markov chains with regular transition matrices are aperiodic.

Example 8.14 *The Markov chain having the following transition matrix has a period of 3.*

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & & \\ 0 & 0 & 1 \\ 0 & & \\ 0 & 0 & 0 \\ 1 & & \\ 0.1 & 0.9 & 0 \\ 0 & & \end{bmatrix}$$

The Markov chain path runs from states 1 or 2 to state 3 and then state 4 before returning to states 1 or 2, and so on.

Example 8.15 *The Markov chain having the following transition matrix has a period of 1.*

$$P = \begin{bmatrix} .1 & 0 & .9 \\ 0 & & \\ 0 & 0 & 1 \\ 0 & & \\ 0 & 0 & 0 \\ 1 & & \\ 0.1 & 0.9 & 0 \\ 0 & & \end{bmatrix}$$

The possibility of repeated visits to state 1 makes this Markov chain regular, and hence aperiodic. You can see that P is regular by calculating P^6 and noting that all entries are nonzero.

8.3.4 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.

8.3.5 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] 9.28136

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

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

The two results match.

8.3.6 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 $P_{ij}^{(n)} > 0$.

Example 8.16 *Mouse maze: State 1 leads to state 2; State 2 leads to state 1, and so on.*

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

Example 8.17 *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.

Example 8.18 *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

Example 8.19 *Mouse Maze e.g.: All states communicate. They form a class.*

Example 8.20

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

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.

Example 8.21 *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.

Example 8.22 *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.

Example 8.23 P_{33} *is aperiodic.*

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

Example 8.24 P_{33} *is regular.* P *for the mouse maze example is not regular.*

8.3.7 Calculation of steady state vector

The steady state vector $\mathbf{q}$ solves

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

which can be re-written as

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

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

Example 8.25
```
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
```

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 steady state vector is a (discrete) probability distribution, its elements must sum to 1:

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

```
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
```

8.3.8 Solution of Linear Systems via QR

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.

8.3.9 Mouse Odor Example

Recall the mouse odor model:

```
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
A <- t(P) - diag(rep(1, 4))  # P^T - I
A <- rbind(A, rep(1,4))  # additional row
RHS <- c(rep(0,4), 1)
qr.solve(A, RHS)
```

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

Example 8.26 *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* t*th day.*
2. *Find the long run 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

$$\textit{Profit} = 100 \times X$$

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. Show that P_5, the transition matrix for the completely random mouse maze example with five compartments, is a regular matrix. What is the period of this Markov chain? Find the steady state distribution for the Markov chain. Is the simulation result that was obtained in Section 8.1.5 consistent with this?

8.4 Markov Chain Monte Carlo simulation

```
set.seed(222696)   # use this to reproduce output
```

8.4.1 Reversible Markov Chains

A Markov chain is said to be *time-reversible* if the Markov property holds for the chain when it is time reversed:

$$P(X_n = x_n | X_{n+1} = x_{n+1}, \ldots) = P(X_n = x_n | X_{n+1} = x_{n+1})$$

A Markov Chain with transition matrix P is reversible if there exists a vector $\mathbf{q}$ such that

$$q_j P_{ji} = q_i P_{ij}.$$

For such a $\mathbf{q}$, observe what happens when we multiply $\mathbf{q}$ 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 $\mathbf{q}$ is the steady state vector for P.

Example 8.27 *Symmetric Random Walk*
Suppose $S = \{0, \pm 1, \pm 2, \ldots, \pm k\}$ for some $k > 2$.

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

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 show that this Markov chain is time-reversible.
For $|j| < k$,

$$P_{j,j+1} = P_{j,j-1} = 0.5.$$

and

$$P_{k,k-1} = 1 = P_{-k,-k+1}.$$

$$q_k P_{k,k-1} = q_{k-1} P_{k-1,k} \quad or$$
$$q_k = q_{k-1} \times 0.5$$

and similarly,

$$q_{-k} = q_{1-k} \times 0.5.$$

For $|j| < k-1$,

$$q_j P_{j,j+1} = q_{j+1} P_{j+1,j} \quad or$$
$$0.5 q_j = 0.5 q_{j+1}.$$

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 8.28 *For the previous example, the steady-state distribution is given by*

$$q_j = \frac{1}{2(k-1)+1+1}$$

and

$$q_k = q_{-k} = \frac{1}{4k}$$

For the special case where $k = 3$, *we have*

$$q_3 = q_{-3} = \frac{1}{12}$$

$$q_2 = q_1 = q_0 = q_{-1} = q_{-2} = \frac{1}{6}.$$

A simulation check on the previous calculation

For the preceding example, the transition matrix is given by

```
P <- matrix(c(0,1,0,0,0,0,0,
.5,0,.5,0,0,0,0,0,.5,0,.5,0,0,0,
0,0,.5,0,.5,0,0,0,0,0,0.5,0,.5,0,
0,0,0,0,.5,0,.5,0,  0,  0,  0,  0,  1,  0), nrow=7, byrow = TRUE)
```

Simulating the random walk is as follows:

```
Ntransitions <- 100000 # number of moves
location <- numeric(Ntransitions)#initializing
current.state <- 1 # initial stock
for (t in 1:Ntransitions) {
   current.state <- sample(1:7,
        size = 1, prob = P[current.state, ])
   location[t] <- current.state
}
pi <- table(location)/Ntransitions
pi

## location
##       1       2       3       4       5       6       7
## 0.08196 0.16490 0.16702 0.16906 0.16881 0.16604 0.08221
```

Conventional calculation of the steady-state vector

```
A <- t(P) - diag(rep(1,7))
A <- rbind(A, rep(1,7))
RHS <- c(rep(0,7), 1)
options(digits=3)
qr.solve(A, RHS)

## [1] 0.0833 0.1667 0.1667 0.1667 0.1667 0.1667 0.0833
```

8.4.2 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 set to ensure that the row sums of P are 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.

8.4.3 Simulating from the Infinite State Markov Chain

Example:

Suppose $\pi_i = k/(i+1)^4$ for $i > 0$ and $\pi_i = 0$ for all $i < 1$. k is a constant that ensures that $\sum_{i=1}^{\infty} \pi_i = 1$. Note that we can simulate from this Markov chain even without knowing k.

```
pi.fun <- function(i) {
    out <- 0
    if (i > 0) out <- 1/(i+1)^4
    out
}
```

8.4.4 Simulating from the Infinite State Markov Chain

```
Ntransitions <- 20000
X <- numeric(Ntransitions)
current.state <- 50  # initialize the Markov chain
for (n in 1:Ntransitions) {
     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)])
```

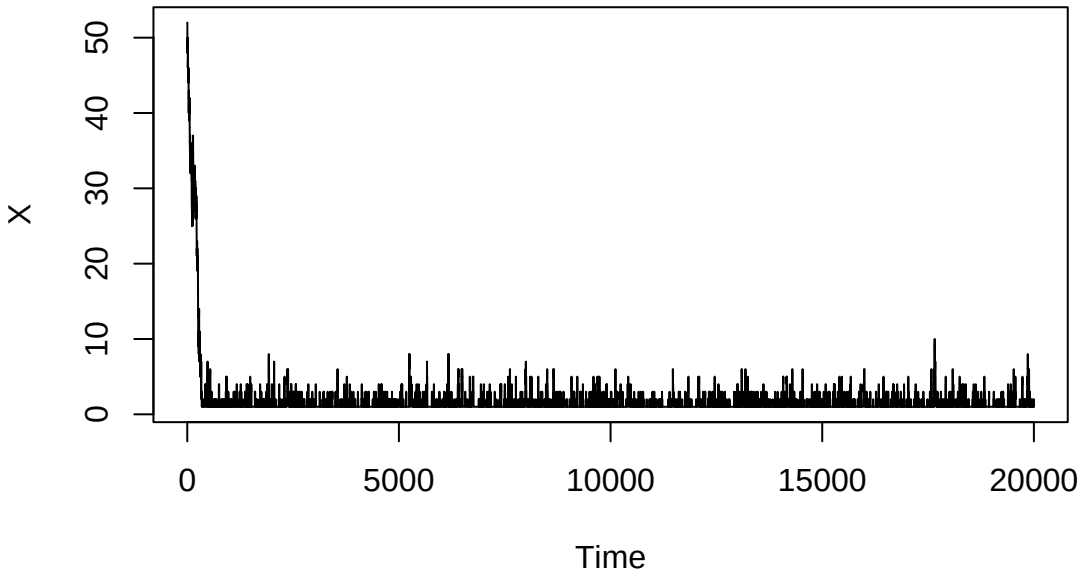
8.4.5 *Simulating from the Infinite State Markov Chain*

```
observedDist

##
##     1     2     3     4     5     6     7     8     9    10
## 14662  2865   846   334   160    94    18    14     3     4
```

8.4.6 *Burn-In*

Why omit the first 1000 observations?

```
ts.plot(X)
```



8.4.7 *Estimating* k

$$\pi_2 = k/(2+1)^4 = k/81$$

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
## 12.2
```

This procedure is one version of MCMC – developed by Metropolis and Hastings.
Procedure:

1. Given a distribution π, known up to a proportionality constant (k), find a time-reversible Markov chain with π as the steady state vector.
2. Simulate from that Markov chain.
3. After simulating for a long enough period (burn-in), the observed states follow the steady state distribution, i.e. π.

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

$$\frac{1}{N}\sum_{n=1}^{N} X_n \text{ and } \frac{1}{N}\sum_{n=1}^{N} g(X_n).$$

8.4.8 *MCMC Application - Bayesian Statistics*

Example:

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 $X = 5$. Use MCMC to simulate the distribution of N, given X.

Terminology: the Poisson distribution for N is the *prior* distribution.

the distribution of N, given $X = 5$, is called the *posterior* distribution.

```
posterior.fun <- function(i, x) {
    out <- 0
    if (i >= x) out <- dpois(i, lambda = 20)*
              dbinom(x, size = i, prob = .5)
    out
}
```

```
pi.fun <- function(i) {
    posterior.fun(i, x=5)
}
```

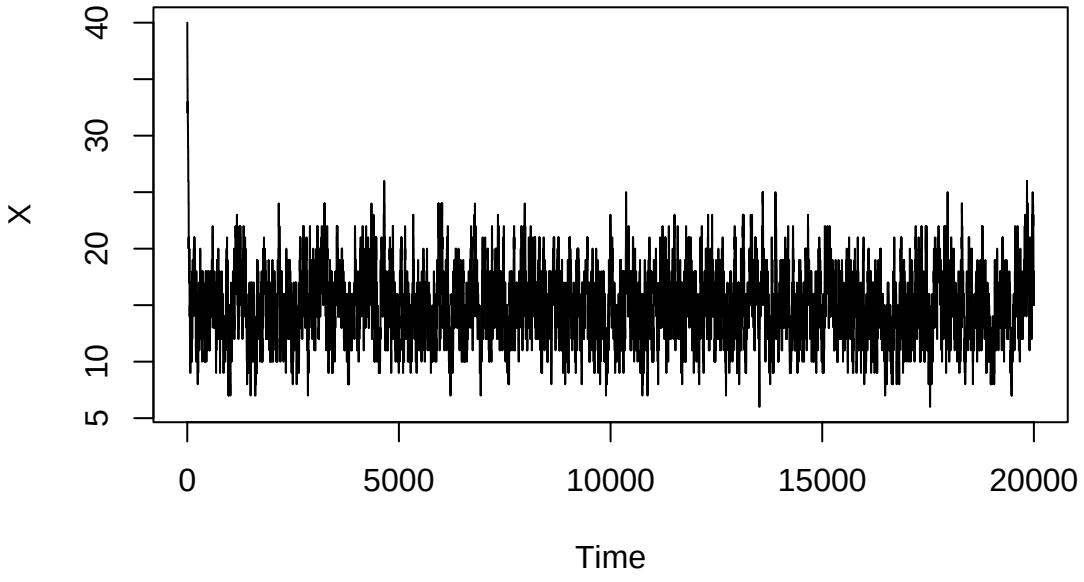
Simulating the Markov chain

```
Ntransitions <- 20000
X <- numeric(Ntransitions)
current.state <- 40  # initialize the Markov chain
for (n in 1:Ntransitions) {
     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)])
```

Plotting the Trace

```
ts.plot(X)
```

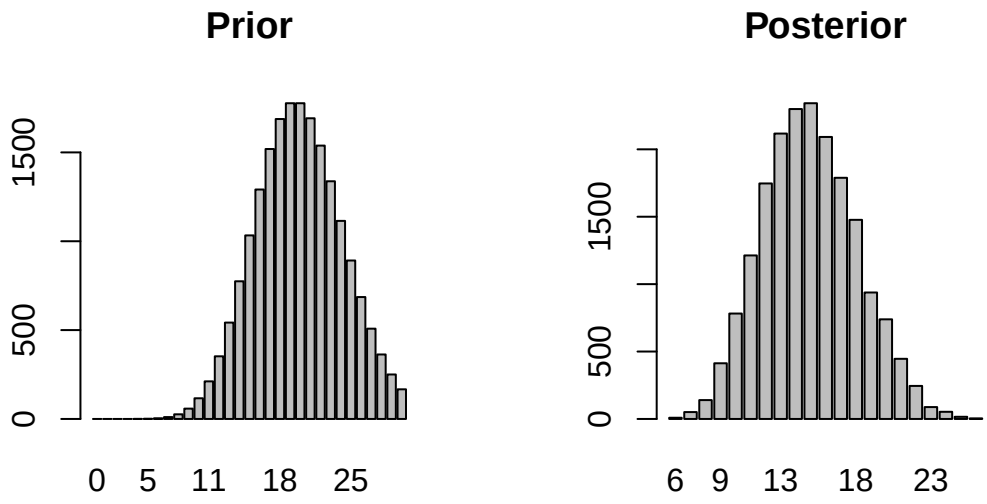


Posterior distribution of N

```
options(width=50)
observedDist
```

```
##
##    6    7    8    9   10   11   12   13   14   15
##    9   51  140  413  782 1213 1747 2117 2299 2342
##   16   17   18   19   20   21   22   23   24   25
## 2092 1789 1477  938  739  446  245   88   53   16
##   26
##    4
```

```
par(mfrow=c(1,2))
theoryDist <- 20000*dpois(0:30, lambda = 20)
names(theoryDist) <- 0:30
barplot(theoryDist, main = "Prior")
barplot(observedDist, main = "Posterior")
```



This is how the *data* $X = 5$ influences our *belief* (initially, Poisson(20)) about the distribution of the unknown value N.

8.4.9 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
}
```

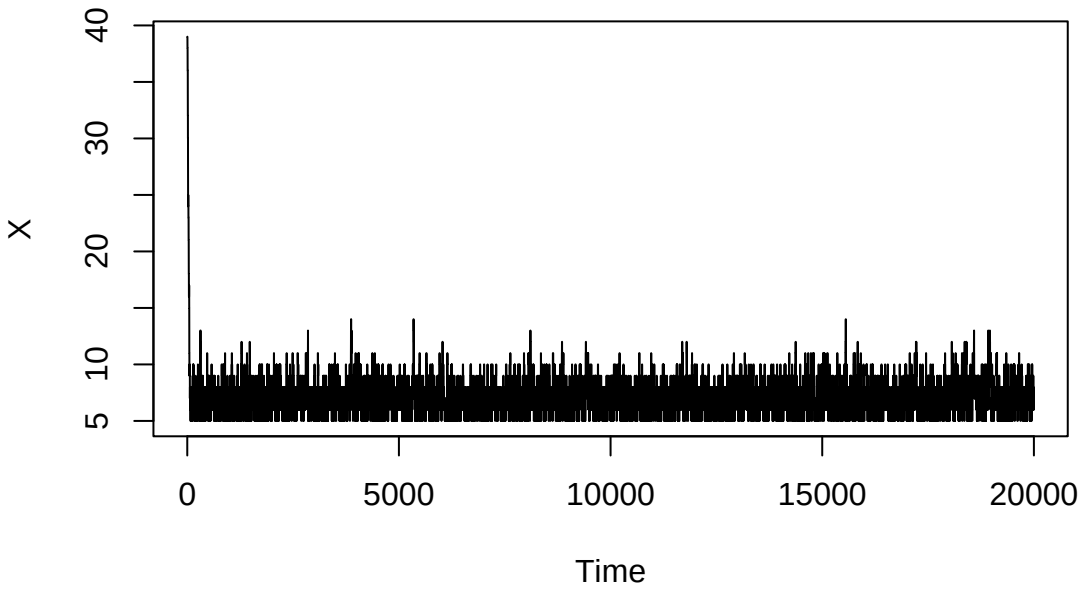
Simulating the Markov chain

```
Ntransitions <- 20000
X <- numeric(Ntransitions)
current.state <- 40  # initialize the Markov chain
for (n in 1:Ntransitions) {
     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)])
```

Plotting the trace

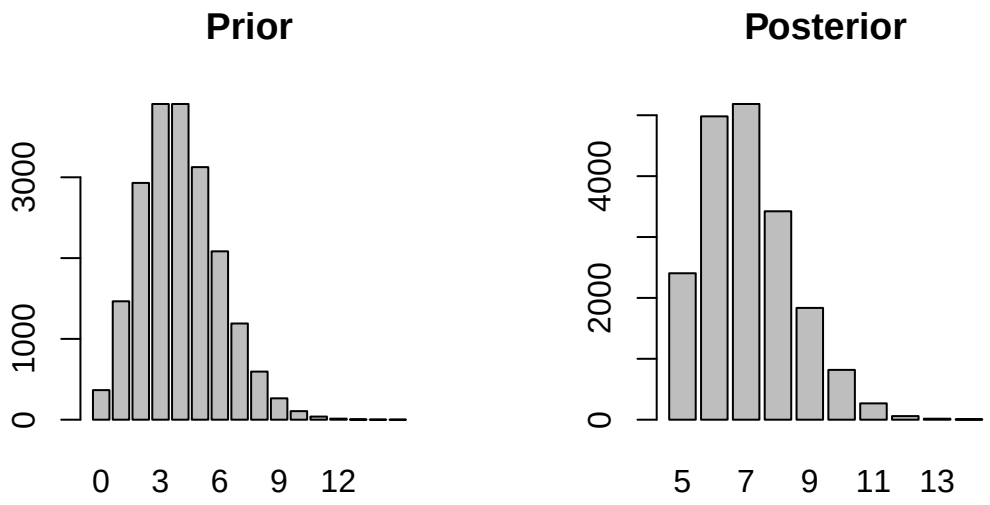
```
ts.plot(X)
```



Posterior distribution of N

```
options(width=50)
observedDist

##
##    5    6    7    8    9   10   11   12   13   14
## 2405 4982 5185 3423 1836  818  268   59   16    8
```

```
par(mfrow=c(1,2))
theoryDist <- 20000*dpois(0:15, lambda = 4)
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.

8.4.10 Using built-In software

Perhaps the best way to do MCMC in R is with the `metrop()` function in C. Geyer's *mcmc* package:

```
metrop(obj, initial, nbatch, blen = 1, nspac = 1,
   scale = 1, outfun, debug = FALSE, ...)
```

Main Arguments:

- `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.

8.5 Hidden Markov models

```
set.seed(222696)   # use this to reproduce output
```

8.5.1 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*, and we can form them

into a matrix E, whose (i, j) entry is the conditional probability that $Y = j$, given $X = i$. In this case,

$$E = \begin{bmatrix} .75 & .25 \\ .25 & .75 \end{bmatrix}.$$

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 E or the transition probabilities P. These are usually estimated by maximizing the likelihood function:

$$L(E, P) = P(Y_1, Y_2, \ldots, Y_n).$$

Because the Y_j's are not independent, we need to be careful how to evaluate the likelihood.

What we can calculate easily is:

$$P(Y_1, Y_2, \ldots, Y_n | X_1, X_2, \ldots, X_n) = \Pi_{j=1}^{n} (.25 + .5X_j)^{Y_j} (.75 - .5X_j)^{1-Y_j}.$$

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1) \cdots P(X_n|X_{n-1})$$

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.

8.5.2 Dynamic programming

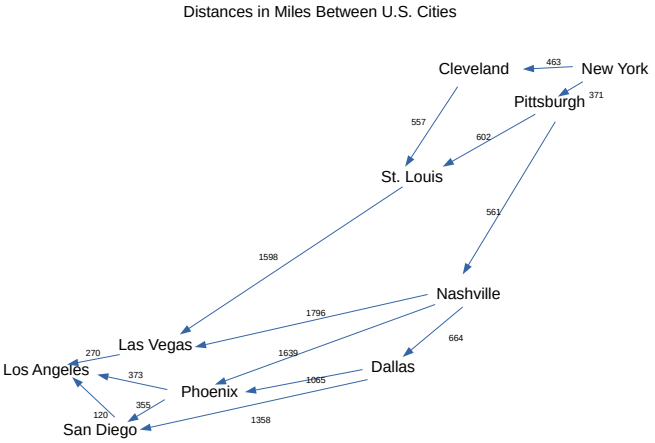
Dynamic programming is an optimization procedure due to Richard Bellman (early 1960's) which efficiently finds the minimum distance route through a network. Consider the problem of driving from Los Angeles to New York. There are many possible routes through the network of cities in between. Dynamic programming allows us to break the bigger network problem into a set of smaller network problems which can be solved recursively.

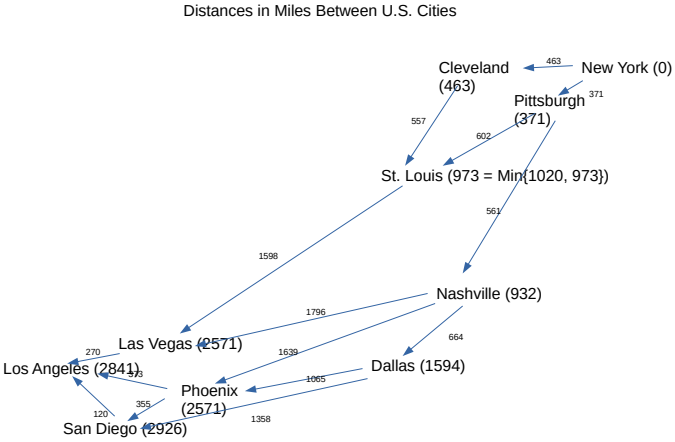
8.5.3 The Basic Idea of the Viterbi Algorithm

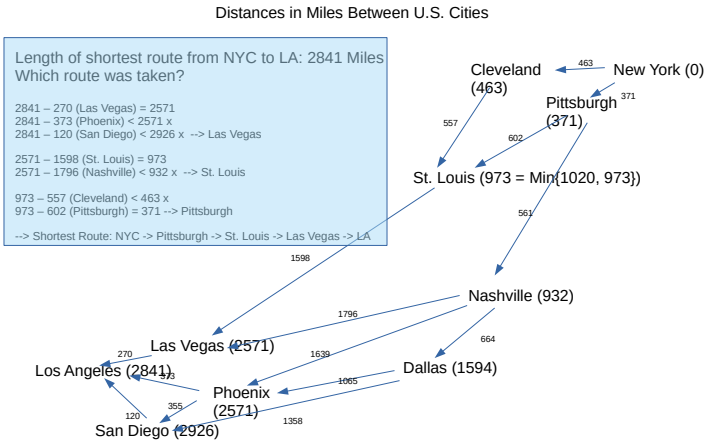
Dynamic programming can also be used to maximize quantities through a network. The Viterbi algorithm employs dynamic programming to find the mostly likely sequence of hidden Markov chain states which could give rise to the observed data. Using the Viterbi path, we can calculate a kind of maximum likelihood 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.

Example 8.29 ***Find Shortest Route from New York to Los Angeles***


Distances in Miles Between U.S. Cities
Cleveland
New York
Pittsburgh
St. Louis
Nashville
Las Vegas
Los Angeles
Phoenix
Dallas
San Diego
463
371
557
602
561
1598
1796
664
270
1639
373
1065
355
120
1358


Distances in Miles Between U.S. Cities
Cleveland (463)
New York (0)
Pittsburgh (371)
St. Louis (973 = Min{1020, 973})
Nashville (932)
Las Vegas (2571)
Los Angeles (2841)
Phoenix (2571)
Dallas (1594)
San Diego (2926)
463
371
557
602
561
1598
1796
664
270
1639
373
1065
355
120
1358


Distances in Miles Between U.S. Cities
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
Cleveland (463)
New York (0)
Pittsburgh (371)
St. Louis (973 = Min{1020, 973})
Nashville (932)
Las Vegas (2571)
Los Angeles (2841)
Phoenix (2571)
Dallas (1594)
San Diego (2926)
463
371
557
602
561
1598
1796
664
270
1639
373
1065
355
120
1358

8.5.4 Built-in Software - the depmixS4 package

The function `depmix()` can be used to fit general purpose Hidden Markov Models. Its use goes beyond the scope of this book. Instead, we focus on a simpler-to-use package, *HMM*, 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.

```
library(HMM)
```

```
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))
```

Using the `viterbi` function, we can calculate the Viterbi path, the most likely X values:

```
viterbi <- viterbi(hmm, observations)
print(viterbi)

## [1] "1" "0" "1" "0" "1" "1" "0"
```

8.5.5 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.

8.5.6 An Example Using Simulated Data

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 $X = 1$.

```
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

```
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.

```
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:

```
print(simDataFit$hmm$transProbs)  # transition matrix estimate

##      to
## from     0     1
##    0 0.128 0.872
##    1 0.202 0.798
```

```
print(simDataFit$hmm$emissionProbs)  # emission matrix estimate

##        symbols
## states 0     1     2
##      0 1 0.000 0.000
##      1 0 0.441 0.559
```

The estimated matrices are not terribly 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
```

8.5.7 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 E:

$$E[i, j] = P(Y = j|X = i)$$

where X is the current state of the hidden Markov chain, and Y is the current observation.

e.g. $E[2, 3] = 0.3$ 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.

8.5.8 *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
```

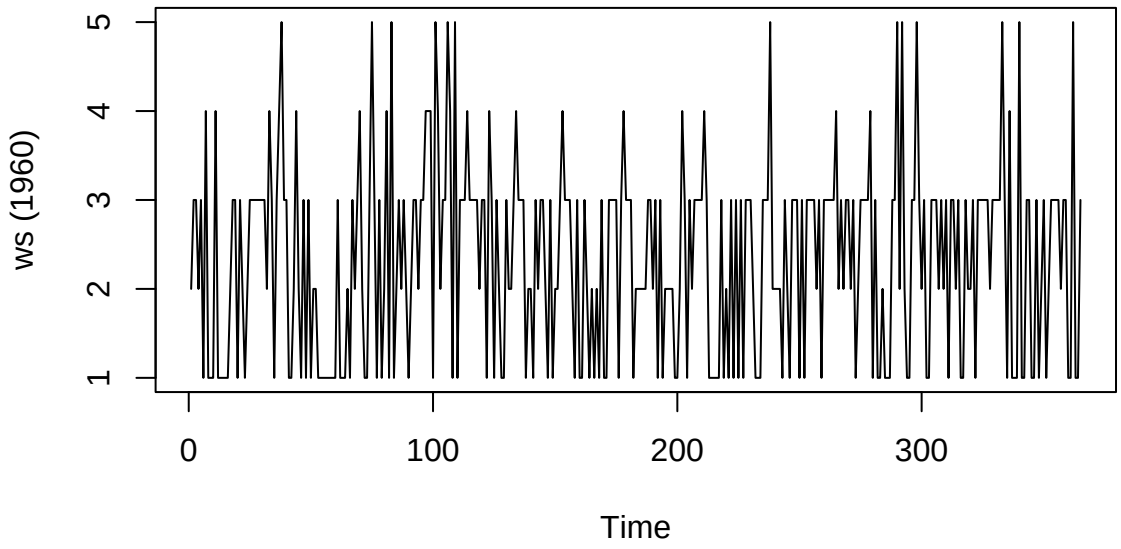
```
table(rle(WindSim$observation))
```

```
##         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.

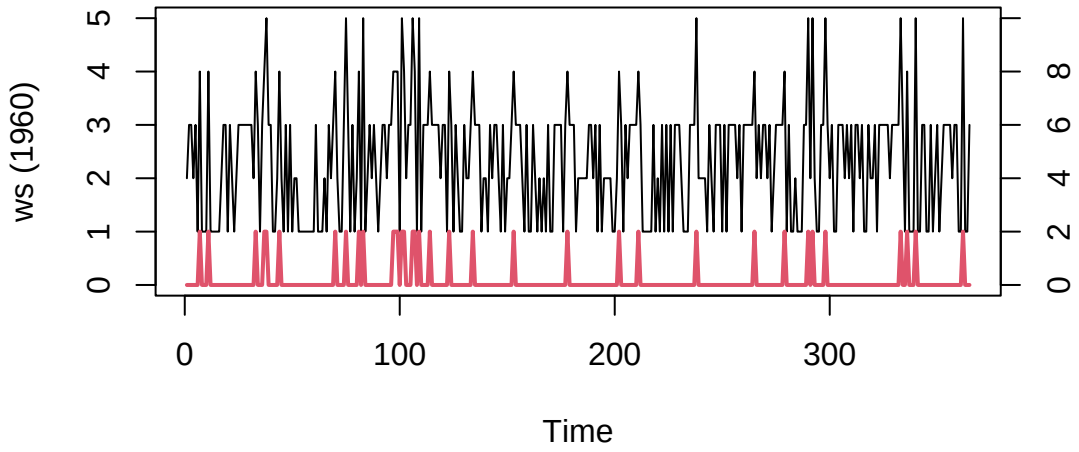
One year of the original data:

```
par(mar=c(4, 4, 1, 1))
ts.plot(WindStates[1:365], ylab="ws (1960)")
```



Including information on the most likely hidden states:

```
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. The data in `DMC.R` concern the duff moisture code which is an indicator of how ready a forested area is to ignite into a wildfire. Values above 20 indicate considerable dryness while values below 10 are not conducive to fire starts.

   (a) Fit a hidden Markov model with 4 hidden states to this data set, by first classifying the DMC values into a set of 3 discrete states as follows:

   ```
   source("DMC.R")
   DMCStates <- cut(DMC, c(-.1, 14, 21, 90))
   ```

(b) Initialize the hidden Markov model using a 4×4 transition matrix and a 4×3 emissions probability matrix.
(c) Fit the HMM using the Viterbi algorithm.
(d) Simulate data from the fitted model.
(e) Compare run lengths for the three states in the simulation with the run lengths in the original data.

2. Fit another HMM to the DMC data, this time using more possible observed states as in:

```
DMCStates <- cut(DMC, c(-.1,  7, 14, 21, 32,  90))
```

(a) Initialize the hidden Markov model using a 4×4 transition matrix and a 4×5 emissions probability matrix.
(b) Fit the HMM using the Viterbi algorithm.
(c) Simulate data from the fitted model.
(d) Compare run lengths for the three states in the simulation with the run lengths in the original data.

3. One way to increase the length of time that a hidden Markov model will stay in one of the observed states is to increase the number of hidden states that have high probability of emission to the states in question. That could lead to stability problems - and computational problems. On the other hand, by decreasing the number of observed states, we could achieve the same goal. This time, use only two states as follows:

```
DMCStates <- cut(DMC, c(-.1, 32, 90))
```

(a) Initialize the hidden Markov model using a 4×4 transition matrix and a 4×2 emissions probability matrix.
(b) Fit the HMM using the Viterbi algorithm.
(c) Simulate data from the fitted model.
(d) Compare run lengths for the three states in the simulation with the run lengths in the original data.

8.6 Continuous-Time Markov chains

```
set.seed(222696)  # use this to reproduce output
```

8.6.1 Some properties of exponential random variables

Memoryless property

If X is an exponential random variable, then for any $r, s \geq 0$,

$$P(X > r + s | X > r) = P(X > s).$$

Example 8.30 *Consider an exponential with $\lambda = 2$, and take $r = 1.5$ and $s = 0.75$:*

```
N <- 1000000 # number of simulated values
r <- 1.5; s <- 0.75
X <- rexp(N, rate = 2)
mean(X[X > r] > (r + s)) # P(X > r+s|X > r)

## [1] 0.2224

mean(X > s) # P(X > s)

## [1] 0.2226
```

Minimum Property I

If X and Y are independent exponential random variables with means $1/\lambda$ and $1/\mu$, respectively. Then

$$P(\min(X, Y) \geq t) = e^{-(\mu+\lambda)t}$$

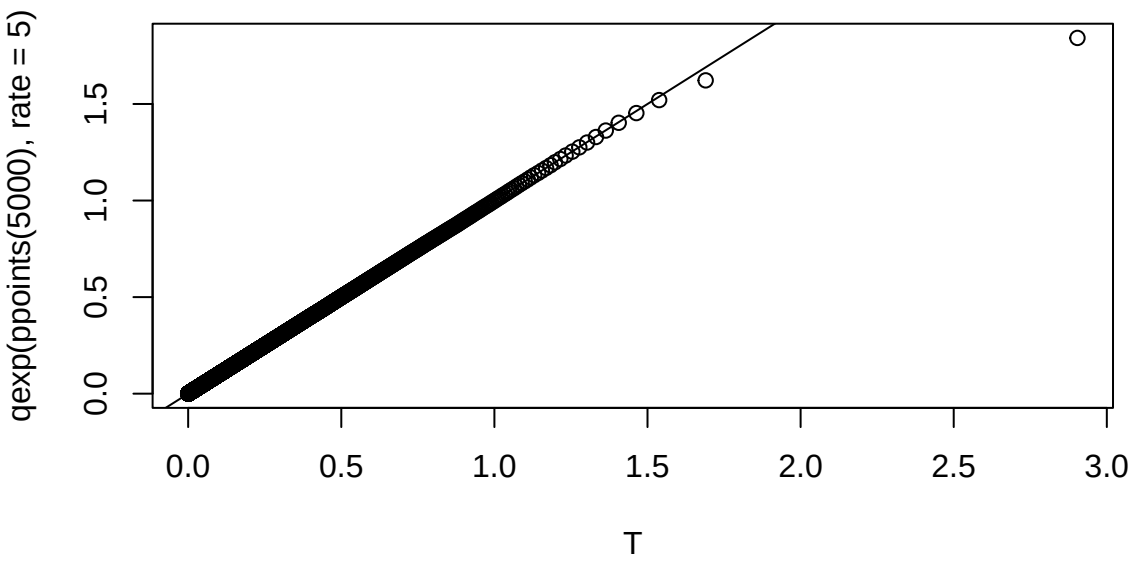
That is, $\min(X, Y)$ is an exponential random variable with mean $\frac{1}{\mu+\lambda}$.

Example 8.31 *Suppose Y has parameter $\lambda = 2$, and X has parameter $\mu = 3$:*

```
Y <- rexp(N, rate = 3);    T <- pmin(X, Y)   # pairwise min
mean(T) # compare with 1/5

## [1] 0.1998
```

An exponential QQ-plot of T is below:

```
par(mar=c(4, 4, 1, 1))
qqplot(T, qexp(ppoints(5000), rate = 5))
abline(0, 1)
```



The points line up along the reference line indicating that T has an exponential distribution with rate 5, as predicted by the property.

Minimum Property II

If X and Y are independent random variables with means $1/\lambda$ and $1/\mu$, respectively, then

$$P(X < Y) = \frac{\lambda}{\lambda + \mu}$$

e.g.

```
mean(X < Y) # compare with 2/5

## [1] 0.4
```

8.6.2 The general framework

We define $X(t)$ as a continuous time Markov chain, if it is a random function of t which is piecewise constant with jumps at random times ... which will be defined presently.

There exist constants q_{jj} and q_{ij} such that for very small Δ, the following hold:

$$P(X(t+\Delta) = j | X(t) = j) = 1 - \Delta q_{jj} + o(\Delta)$$

$$P(X(t+\Delta) = j | X(t) = i) = \Delta q_{ij} + o(\Delta)$$

The term $o(\Delta)$ is very small – if you divide by Δ and let Δ tend to 0, you still get 0.

The first equation essentially says that there is a very high probability of no jump within a short time interval. The second equation says that there is a small probability of a jump from state i to state j in a short time interval.

Let $p_j(t) = P(X(t) = j)$ for $j \in \{1, 2, \ldots, K\} = S$, the state space. Then

$$p_j(t+\Delta) = P(X(t+\Delta) = j) = \Delta \sum_{i \neq j} q_{ij} p_i(t) - \Delta q_{jj} p_j(t) + p_j(t) + o(\Delta).$$

Subtracting $p_j(t)$ from both sides, dividing by Δ and taking the limit as $\Delta \to 0$ gives

$$p_j'(t) = \sum_{i \neq j} q_{ij} p_i(t) - q_{jj} p_j(t).$$

If we define the *infinitesimal generator* Q as the $K \times K$ matrix where $Q_{jj} = -q_{jj}$ and $Q_{ij} = q_{ij}$ for $i \neq j$, we can write

$$P'(t) = P(t)Q$$

where $P(t)$ is a K-vector of the probabilities $p_j(t), j = 1, 2, \ldots, K$. In fact, $P(t)$ could be taken to be a $K \times K$ matrix itself, if we want to take into account the initial value of the Markov chain $X(0)$.

The solution to the matrix differential equation is

$$P(t) = P(0)\mathrm{e}^{Qt}.$$

If the eigenvalues of Q are distinct, we can decompose Q as XDX^{-1}, where X is the matrix of eigenvectors of Q. D is the diagonal matrix of eigenvalues of Q. Then we can write

$$P(t) = P(0)X\mathrm{e}^{Dt}X^{-1}$$

which is much easier to work with.

If the Markov chain has a steady state distribution, the probability distribution should become constant eventually. This means that the derivative of $P(t)$ must become 0. Therefore, the steady state distribution π, if it exists, satisfies

$$0 = \pi Q$$

Therefore, we can find the steady state distribution from the above equation, with the additional condition that the probabilities sum to 1.

What are the q_{ij} and q_{jj} values? The q_{ij} values are determined as exponential rates of transition from state i to state j. The q_{jj} values represent the time that the Markov chain stays in state j before making any other transition. From the fact that $P'(t) = P(t)Q$, we have that $P(t)Q1 = 0$, since the derivative of the sum of the probabilities must be 0. This implies that $Q1 = 0$. That is the row sums of Q must be 0. Therefore, $q_{jj} = 1 - \sum_{j \neq i} q_{ji}$

8.6.3 A simple continuous time Markov chain

Consider a light bulb which lasts for an exponential λ_0 amount of time before burning out. It is replaced in an exponential λ_1 amount of time, before burning out again, and so on. Let

$$X(t) = \begin{cases} 1, & \text{if the bulb works at } t \\ 0, & \text{if the bulb is being replaced at } t \end{cases}$$

and let $P_{0i}(t) = P_0(X(t) = i)$, for $t \geq 0$, $i = 0, 1$. It follows that $P_{00}(0) = 1$, and $P_{01}(0) = 0$.

The goal is to calculate $P_{00}(t)$ for any $t > 0$. Using the memoryless property of the exponential distribution, we can show that

$$P_{00}'(t) = -\lambda_0 P_{00}(t) + \lambda_1 P_{01}(t).$$

Similarly,

$$P'_{01}(t) = \lambda_0 P_{00}(t) - \lambda_1 P_{01}(t).$$

Therefore, we can write

$$P'(t) = PQ$$

where

$$Q = \begin{bmatrix} -\lambda_0 & \lambda_0 \\ \lambda_1 & -\lambda_1 \end{bmatrix} \quad \text{and} \quad P(t) = [P_{00}(t) \;\; P_{01}(t)].$$

The solution of this matrix differential equation is

$$P(t) = P(0)e^{Qt}.$$

If $Q = XDX^{-1}$ for some diagonal matrix D, then

$$P(t) = P(0)Xe^{Dt}X^{-1}.$$

In this case,

$$X = \begin{bmatrix} 1 & \lambda_0/\lambda_1 \\ 1 & 1 \end{bmatrix}$$

and

$$D = \begin{bmatrix} 0 & 0 \\ 0 & -\lambda_0 - \lambda_1 \end{bmatrix}$$

We conclude that

$$P_{00}(t) = \frac{\lambda_1 + \lambda_0 e^{-(\lambda_0+\lambda_1)t}}{\lambda_1 + \lambda_0}$$

If $t \to \infty$, we have

$$q_0 = \frac{\lambda_1}{\lambda_0 + \lambda_1}$$

The limiting steady-state probability vector q can more simply be obtained by solving

$$qQ = 0 \quad q\mathbf{1} = 1$$

8.6.4 *A compartmental model*

Suppose an individual can be in one of 4 states: Healthy, Disease$_1$, Disease$_2$ and Disease$_3$. None of the diseases are immediately fatal and it is possible to recover one's health from any of the three diseases. We suppose that the individual can transit from the healthy state to Disease$_1$ at a rate of 0.1 and to Disease$_2$ with a rate of 0.2. They do not transit directly to Disease$_3$. They can recover from Disease$_1$ at a rate of 0.9 or progress to Disease$_3$ at a rate of 0.1. From Disease$_2$ they can recover at a rate of .4 and progress to Disease$_3$ at a rate of 0.6. From Disease$_3$, they can recover at a rate of 0.1. Write down the Q matrix for this model.

$$Q = \begin{bmatrix} -.3 & .1 & .2 & 0 \\ .9 & -1 & 0 & .1 \\ .4 & 0 & -1 & .6 \\ .1 & 0 & 0 & -.1 \end{bmatrix}.$$

Find the eigenvalues and eigenvectors of Q and hence, the vector of probabilities of being in any of the 4 states at a given time, assuming that the individual starts in the healthy state.

```
Q <- matrix(c(-.3, .9, .4, .1, .1, -1, 0, 0, .2, 0, -1, 0, 0, .1, .6, -.1), nrow=4)
evals <- eigen(Q)$values
evals
```

```
## [1] -1.180e+00 -1.000e+00 -2.204e-01 -5.551e-17
```

```
X <- eigen(Q)$vector
X
```

```
##          [,1]       [,2]     [,3] [,4]
## [1,]  0.18478 -1.046e-16 -0.59722 -0.5
## [2,] -0.91650  8.944e-01 -0.62585 -0.5
## [3,] -0.35439 -4.472e-01  0.07528 -0.5
## [4,] -0.01712 -2.538e-16  0.49596 -0.5
```

```
P <- function(t)  {
    Pmat <- matrix(0, nrow=length(t), ncol=4)
    for (i in 1:length(t)) {
        Pmat[i, ] <- exp(evals*t[i])%*%diag(X[1,])%*%solve(X)
    }
    Pmat
}
```

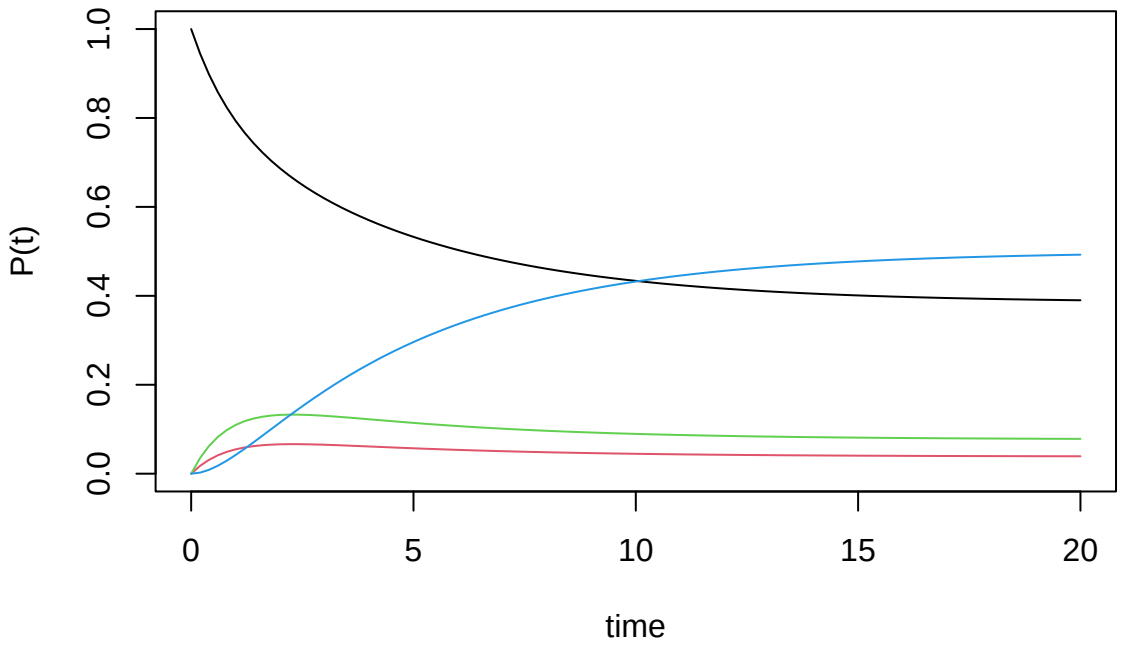
The result is the probabilities which have been calculated as $P(0)\mathrm{e}^{Qt} = P(0)X\mathrm{e}^{Dt}X^{-1}$. The first one (for the healthy state) is $P_1(t) = 0.171\mathrm{e}^{-1.18t} + 0.444\mathrm{e}^{-0.22t} + 0.385$.

Evaluate the probabilities at times 1 and 2, and plot the probabilities as functions of time over the interval from 0 through 20.

```
P(c(1, 2))
```

```
##        [,1]   [,2]  [,3]   [,4]
## [1,] 0.793 0.0548 0.110 0.0421
## [2,] 0.687 0.0661 0.132 0.1152
```

```
par(mar=c(4, 4, 1, 1))
curve(P(x)[,1], 0, 20, xlab="time", ylim=c(0, 1), ylab="P(t)")
for (i in 2:4) {
     curve(P(x)[,i], 0, 20, add = TRUE, col=i)
}
```



8.6.5 *The linear birth process*

In this process, $X(t)$ counts up the number of organisms alive at time t. $X(0)$ is the initial population size and growth of the population increases by 1 each time one of the organisms divides into two offspring. Each living organism does this after being alive for an exponential λ amount of time. For this process, the Q matrix is infinite with nonzero entries at $(j-1, j)$ and (j, j) only: $q_{j-1,j} = \lambda(j-1)$ and $q_{j,j} = -\lambda j$.

It is then possible to show that

$$p_1'(t) = -\lambda p_1(t) \quad \text{and}$$
$$p_j'(t) = \lambda(j-1)p_{j-1}(t) - \lambda j p_j(t).$$

It is easy to solve the first equation to see that

$$p_1(t) = p_1(0)\mathrm{e}^{-\lambda t}.$$

If the initial population size is 1, then $p_1(0) = 1$. The rest of the equations can be solved sequentially, inductively, to obtain

$$p_k(t) = \mathrm{e}^{-\lambda t}(1 - \mathrm{e}^{-\lambda t})^{k-1}, \quad \text{for } k = 2, 3, \ldots.$$

From this, or from the differential equations directly, we can obtain

$$E[X(t)] = X(0)\mathrm{e}^{\lambda t}$$

This is the classical exponential growth model.

Now, suppose that organisms can die as well, and that they live for an exponential μ amount of time before dying. They still divide after an exponential λ amount of time as well. (The memoryless property of exponential random variables helps to resolve the apparent contradiction.) The equations for the probabilities are:

$$p_j'(t) = \lambda(j-1)p_{j-1}(t) + \mu(j+1)p_{j+1}(t) - (\lambda+\mu)p_j(t).$$

Solving these equations is possible, using Laplace transforms $\rightsquigarrow$ beyond the scope of this course. We could also construct equations for nonlinear birth and death processes, by replacing the $j+1$ and $j-1$ terms with functions of $j+1$ and $j-1$, such as square roots or reciprocals and so on.

Finding the mean population size is not difficult. Let $x(t) = E[X(t)]$ and multiply the above equations by j and sum over all j to obtain

$$x'(t) = (\lambda - \mu)x(t)$$

which has the solution $x(t) = \mathrm{e}^{(\lambda-\mu)t}$. If $\lambda > \mu$, the population grows exponentially. If $\lambda < \mu$, the population decays exponentially, and dies out. If $\lambda = \mu$, the population also dies out, but the mean population size is $x(0)$ for all t.

8.6.6 The M/M/1 queue

This queue is a simple birth and death process where the birth and death rates are always the same: λ and μ. Customers arrive at the queue according to a Poisson process with rate λ and they are served according to an exponential distribution with rate μ. The equations for the state probabilities are

$$p_j'(t) = \lambda p_{j-1}(t) + \mu p_{j+1}(t) - (\lambda+\mu)p_j(t)$$

and

$$p_1'(t) = -\lambda p_0(t) + \mu p_1(t).$$

Again, transform methods can be used to solve for the state probabilities.

We can find the steady state probability vector, when $\lambda < \mu$:

$$\lambda p_{j-1}(t) + \mu p_{j+1}(t) - (\lambda+\mu)p_j(t) \to 0$$

as $t \to \infty$, so the steady state probabilities π_j satisfy

$$\lambda\pi_{j-1} + \mu\pi_{j+1} - (\lambda+\mu)\pi_j = 0$$

and

$$-\lambda\pi_0 + \mu\pi_1 = 0.$$

The last equation says that $\pi_1 = \lambda\pi_0/\mu$. We can solve the other equations for π_j in terms of π_0. Then we use the fact that the probabilities must sum to 1, to see that

$$\pi_0 = 1 - \frac{\lambda}{\mu}$$

and

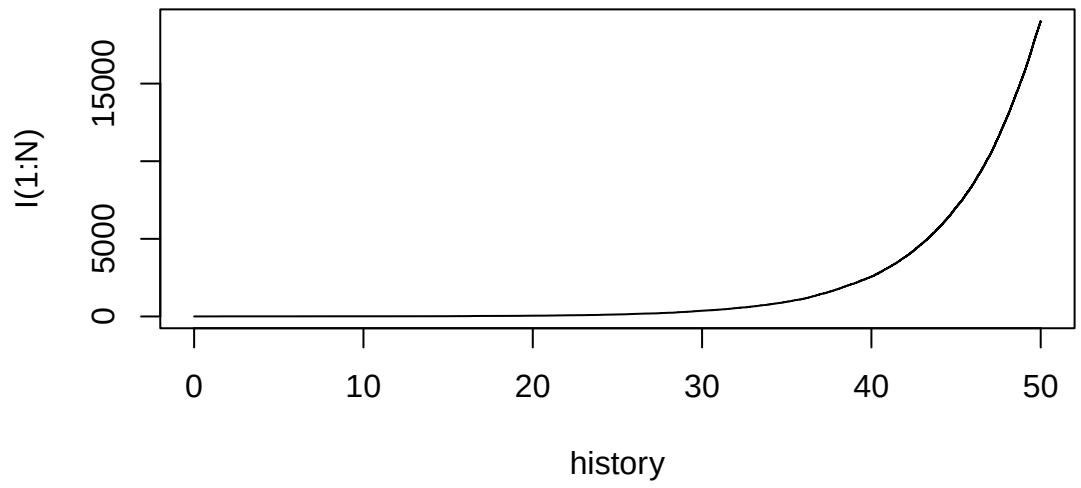
$$\pi_k = \left(\frac{\lambda}{\mu}\right)^k \left(1 - \frac{\lambda}{\mu}\right)$$

for $k = 0, 1, 2, \ldots$.

Exercises

1. Suppose X and Y are independent exponential random variables, each with rate 3, and $Z = \min(X, Y)$. What is the distribution of Z?
2. Suppose the time it takes for a lightbulb to burn out is an exponential random variable with mean 1000 hours, and the time it takes for a second lightbulb to burn out is another independent exponential random variable but with mean 500 hours. What is the probability that the second bulb burns out before the first one?
3. The time it takes for a certain virus to spread from one individual to another is an exponential random variable with mean 50 hours. The lifetime of the virus is exponentially distributed with mean 10 hours. If the virus is dead, it cannot be transmitted. What is the probability that the virus will be transmitted?
4. Suppose cells divide independently of each other according to an exponential distribution with rate λ. (That is, their population grows according to a simple linear birth process.) The code below simulates the growth of such a population, starting from one cell, with $\lambda = .2$, for 50 units of time.

```
N <- 1;  time <- 0 # population at time 0 is 1
history <- time
while (time < 50) {
    X <- rexp(1, rate = .2*N)    # rate is .2 times current population
    # based on minimum property of exponentials
    time <- time + X # time of division
    N <- N + 1 # one cell has divided
    history <- c(history, time)
}
plot(I(1:N) ~ history, type = "l")
```

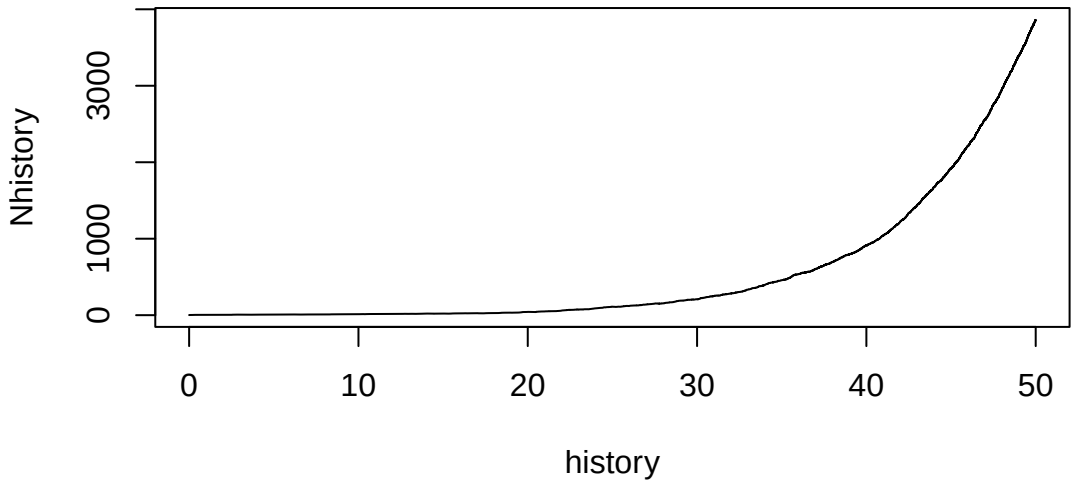


Modify the code so that cell division rate is 0.1. What is the population size at time 50?

5. Refer to the previous exercise. A simple way to incorporate crowding in a birth process is to modify the rate so that it takes into account the current population size. One possibility is to replace λ by λ/N^α, for some α. Modify the code in the previous question so that the rate of division is $0.2/N^{0.1}$ instead of 0.2, and run the simulation for 50 time units and note the population size at time 50.
6. A cell can die before it divides. If the time to cell death is exponentially distributed with rate μ, independently of the other cells, it is possible to modify the code to incorporate this (here we assume $\mu = .06$ and $\lambda = .2$:

```
N <- 1;  time <- 0 # population at time 0 is 1
history <- time; Nhistory <- N
while (time < 50 & N > 0) {
```

```
    X <- rexp(1, rate = (.2 + .06)*N)    # time to birth/death
    # based on minimum property of exponentials
    time <- time + X # time of division or death
    BD <- rbinom(1, 1, .06/(.2 + .06)) # this is 1 if cell dies
    N <- N + 1 - 2*BD # one cell has divided or died
    Nhistory <- c(Nhistory, N)
    history <- c(history, time)
}
plot(Nhistory ~ history, type = "l")
```



Modify the code so that the death rate μ is .05. What happens to the population?

7. Suppose a continuous time Markov chain with 3 states has infinitesimal generator Q:

$$Q = \begin{bmatrix} -1 & .1 & .9 \\ .9 & -1 & .1 \\ 2 & 3 & -5 \end{bmatrix}.$$

Find the steady-state distribution for this Markov chain.

8. Consider a 2 state Markov chain where the rate of transition from state 1 to state 2 is 100 and the rate of transition from state 2 to state 1 is 1. Find Q for this system and calculate the steady-state distribution.
9. Consider a 3 state Markov chain where the rate of transition from state 1 to state 2 is 100, the rate from 1 to 3, is 50, the rate from 2 to 1 is 1 and the rate from 2 to 3 is 50, and the rate from 3 to 1 is 1 and the rate from 3 to 2 is 10. Find Q for this system and calculate the steady-state distribution.

Bibliography

[1] Barrios, E. (2016). BHH2: Useful Functions for Box, Hunter and Hunter II. R package version 2016.05.31. URL https://CRAN.R-project.org/package=BHH2.

[2] Braun, W.J. (2019). MPV: Data Sets from Montgomery, Peck and Vining. R package version 1.55. URL https://CRAN.R-project.org/package=MPV.

[3] Braun, W.J. and Murdoch, D. (2021). *A First Course in Statistical Programming with R* Third Edition. Cambridge University Press.

[4] Dragulescu, A.A. and Arendt, C. (2018). xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files. R package version 0.6.1. URL https://CRAN.R-project.org/package=xlsx.

[5] Hyndman R, Athanasopoulos G, Bergmeir C, Caceres G, Chhay L, O'Hara-Wild M, Petropoulos F, Razbash S, Wang E and Yasmeen F (2018). *forecast: Forecasting functions for time series and linear models*. R package version 8.4, http://pkg.robjhyndman.com/forecast.

[6] Hyndman RJ and Khandakar Y (2008). "Automatic time series forecasting: the forecast package for R." *Journal of Statistical Software*, **26** 1–22. http://www.jstatsoft.org/article/view/v027i03.

[7] Knuth, D.E. (1997) *The Art of Computer Programming Volume 2: Seminumerical Algorithms. Third Edition.* Reading, Massachusetts: Addison-Wesley.

[8] Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. *R News* **2** 18–22.

[9] Ligges, U. and Mächler, M. (2003). Scatterplot3d - an R Package for Visualizing Multivariate Data. *Journal of Statistical Software* **8** 1–20.

[10] Maindonald, J.H. and Braun, W.J. (2006). *Data Analysis and Graphics.* Third Edition. Cambridge University Press.

[11] Maindonald, J.H. and Braun, W.J. (2015). DAAG: Data Analysis and Graphics Data and Functions. R package version 1.22. URL https://CRAN.R-project.org/package=DAAG.

[12] Matsumoto, M. and Nishimura, T. (1998) Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation*, **8** 3–30.

[13] R Core Team (2019). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

[14] R Core Team (1999-2018). An Introduction to R. Version 3.6.1 (2019-07-05).

[15] Sarkar, D.(2008) *Lattice: Multivariate Data Visualization with R.* Springer, New York. ISBN 978-0-387-75968-5

[16] Therneau, T. and Atkinson, A. (2018). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-13. https://CRAN.R-project.org/package=rpart

[17] van Buuren, S. and Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45(3), 1-67. URL https://www.jstatsoft.org/v45/i03/.

[18] Wichmann, B. A. and Hill, I. D. (1982) Algorithm AS 183: An Efficient and Portable Pseudo-random Number Generator, *Applied Statistics*. **31** 188–190; Remarks: **34**,198 and **35** 89.

Index