Use R !

R by Example

Springer

Jim Albert • Maria Rizzo

R by Example

Springer

Jim Albert
Department of Mathematics and Statistics
Bowling Green State University
Bowling Green Ohio
USA
albert@bgsu.edu

Maria Rizzo
Department of Mathematics and Statistics
Bowling Green State University
Bowling Green Ohio
USA
mrizzo@bgsu.edu

Series Editors:
Robert Gentleman
Program in Computational Biology
Division of Public Health Sciences
Fred Hutchinson Cancer Research Center
1100 Fairview Ave. N. M2-B876
Seattle, Washington 98109-1024
USA

Kurt Hornik
Department für Statistik und Mathematik
Wirtschaftsuniversität Wien Augasse 2-6
A-1090 Wien
Austria

Giovanni Parmigiani
The Sidney Kimmel Comprehensive
Cancer Center at Johns Hopkins University
550 North Broadway
Baltimore, MD 21205-2011
USA

ISBN 978-1-4614-1364-6 e-ISBN 978-1-4614-1365-3
DOI 10.1007/978-1-4614-1365-3
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011941603

Preface

R by Example is an example-based introduction to the R [40] statistical computing environment that does not assume any previous familiarity with R or other software packages. R is a statistical computing environment for statistical computation and graphics, and it is a computer language designed for typical and possibly very specialized statistical and graphical applications. The software is available for unix/linux, Windows, and Macintosh platforms under general public license, and the program is available to download from `www.r-project.org`. Thousands of contributed packages are also available as well as utilities for easy installation.

The purpose of this book is to illustrate a range of statistical and probability computations using R for people who are learning, teaching, or using statistics. Specifically, this book is written for users who have covered at least the equivalent of (or are currently studying) undergraduate level calculus-based courses in statistics. These users are learning or applying exploratory and inferential methods for analyzing data and this book is intended to be a useful resource for learning how to implement these procedures in R.

Chapters 1 and 2 provide a general introduction to the R system and provide an overview of the capabilities of R to perform basic numerical and graphical summaries of data. Chapters 3, 4, and 5 describe R functions for working with categorical data, producing statistical graphics, and implementing the exploratory data analysis methods of John Tukey. Chapter 6 presents R procedures for basic inference about proportions and means. Chapters 7 through 10 describe the use of R for popular statistical models, such as regression, analysis of variance (ANOVA), randomized block designs, two-way ANOVA, and randomization tests. The last section of the book describes the use of R in Monte Carlo simulation experiments (Chapter 11), Bayesian modeling (Chapter 12), and Markov Chain Monte Carlo (MCMC) algorithms to simulate from probability distributions (Chapter 13).

One general feature of our presentation is that R functions are presented in the context of interesting applications with real data. Features of the useful R function `lm`, for example, are best communicated through a good regression example. We have tried to reflect good statistical practice through the examples that we present in all chapters. An undergraduate student should easily be able to relate our R work on, say, regression with the regression material that is taught in his statistics course. In each chapter, we include exercises that give the reader practice in implementing the R functions that are discussed.

The data files used in the examples are available in R or provided on our web site. A few of the data files can be input directly from a web page, and there are also a few that found in the recommended packages (installed with R) or contributed packages (installed by the user when needed). The web page for this book is `personal.bgsu.edu/~mrizzo/Rx`.

Remarks or tips about R are identified by the symbol $\mathbf{R_x}$ to set them apart from the main text. In the examples, R code and output appears in bold monospaced type. Code that would be typed by the user is identified by the leading prompt symbol `>`. Scripts for some of the functions in the examples are provided in files available from the book web site; these functions are shown without the prompt character.

The R manuals and examples in the help files use the arrow assignment operators `<-` and `->`. However, in this book we have used the equal sign `=` operator for assignment, rather than `<-`, as novice users may find it easier to type the `=` symbol.

R functions and keywords are collected at the beginning of the Index. Examples are also indexed; see the entry 'Example' in the Index.

Bowling Green, Ohio *Jim Albert*
Maria Rizzo

Contents

Notation and Abbreviations

$\in$	is an element of
$\propto$	is proportional to
$\Gamma(a)$	complete gamma function, $\Gamma(a) = \int_0^\infty t^{a-1} \exp(-t)\, dt$

cdf	cumulative distribution function
csv	comma separated values (file format)
E	expected value
GUI	graphical user interface
iid	independent and identically distributed
IQR	interquartile range
log	natural logarithm (base e)
NID	normally distributed and independent
QQ	quantile-quantile (plot)
MC	Monte Carlo (methods)
MCMC	Markov chain Monte Carlo
M-H	Metropolis-Hastings (algorithm)
MSE	mean squared error
MST	mean square for treatments
SS.total	total sum of squares
SSE	sum of squares for error (residual sum of squares)
SST	sum of squares for treatments

Chapter 1
Introduction

R is a statistical computing environment. It is free (open source) software for statistical computation and graphics [40] and a computer language designed for typical statistical and graphical applications. The R distribution includes the ability to save and run commands stored in script files, and an integrated editor in the R Graphical User Interface (R-GUI). It is available for most platforms including unix/linux, PC, and Macintosh platforms. Thousands of contributed packages are available, and users are provided tools to make packages.

At the core of R is an interpreted computer language. This language provides the logical control of branching and looping, and modular programming using functions. The base R distribution contains functions and data to implement and illustrate most common statistical procedures, including regression and ANOVA, classical parametric and nonparametric tests, cluster analysis, density estimation, and much more. An extensive suite of probability distribution functions and generators are provided, as well as a graphical environment for exploratory data analysis and creating presentation graphics.

On the history and evolution of R, see the R-FAQ [26] and resources on the R home page at `http://www.R-project.org/`.

1.1 Getting Started

R is an interpreted language; that is, the system processes commands entered by the user, who types the commands at the command prompt, or submits the commands from a file called a script. We assume that our readers use R at a graphics workstation running a windowing system, such as Windows, Macintosh, or X window systems. In a window system, users interact with R through the R console. Except for the simplest operations, most users will prefer to type commands in a script (see Section 1.1.3) to save retyping and to separate commands from results. However, let us begin by working directly at the command prompt.

When we use the command line interface, each command or expression to be evaluated is typed at the command prompt, and immediately evaluated when the Enter key is pressed at the end of a syntactically complete statement. It is helpful to remember the following tips.

- Press the up-arrow key to recall commands and edit them.
- Use the Esc (Escape) key to cancel a command.

1.1.1 Preliminaries

Remarks or tips about R are identified by the symbol **Rx** to set them apart from the main text.

Rx 1.1 *The right-to-left assignment operators are the left arrow* `<-` *and equal sign* `=`*. For example, borrowing a line from Example 1.3, either method below*

```
> x = c(109, 65, 22, 3, 1)
> x <- c(109, 65, 22, 3, 1)
```

creates the vector $(109, 65, 22, 3, 1)$ *and assigns it to* `x`*. Borrowing another line from Example 1.3, either method below*

```
> y = rpois(200, lambda=.61)
> y <- rpois(200, lambda=.61)
```

assigns the result of the `rpois` *function to* `y`*. Notice that the equal sign inside the parentheses is not an assignment operator; it passes the value of an argument (lambda) to the function* `rpois`*.*

The R manuals and examples in the help files use the arrow assignment operators `<-` and `->`. However, in this book we have used the equal sign `=` operator for assignment, rather than `<-`, as novice users may find it easier to type the `=` symbol.

In the examples, R code and output appears in bold monospaced type as in the remark **Rx** 1.1 above. Code that would be typed interactively by the user or submitted from an R script is identified by the leading prompt symbol `>`. Scripts for some of the functions in the examples are provided in files available from the book web site; these functions are shown in the book without the prompt character.

Data files and scripts used in the examples are available on our web site at `personal.bgsu.edu/~mrizzo/Rx`. Some data files can be downloaded directly from a connection to a url.

1.1.2 Basic operations

Some basic operations with vectors are illustrated in the following example. The R commands are entered at the prompt in the R console window. The prompt character is > and when a line is continued the prompt changes to +. (The prompt symbols can be changed.)

Example 1.1 (Temperature data). Average annual temperatures in New Haven, CT, were recorded in degrees Fahrenheit, as

```
Year                1968   1969   1970  1971
Mean temperature    51.9   51.8   51.9    53
```

(This data is part of a larger data set in R called *nhtemp*.) The combine function `c` creates a vector from its arguments, and the result can be stored in user-defined vectors. We use the combine function to enter our data and store it in an object named `temps`.

```
> temps = c(51.9, 51.8, 51.9, 53)
```

To display the value of `temps`, one simply types the name.

```
> temps
[1] 51.9 51.8 51.9 53.0
```

Suppose that we want to convert the Fahrenheit temperatures (F) to Celsius temperatures (C). The formula for the conversion is $C = \frac{5}{9}(F - 32)$. It is easy to apply this formula to all of the temperatures in one step, because arithmetic operations in R are *vectorized*; operations are applied element by element. For example, to subtract 32 from every element of `temp`, we use

```
> temps - 32
[1] 19.9 19.8 19.9 21.0
```

Then `(5/9)*(temps - 32)` multiplies each difference by 5/9. The temperatures in degrees Celsius are

```
> (5/9) * (temps - 32)
[1] 11.05556 11.00000 11.05556 11.66667
```

In 1968 through 1971, the mean annual temperatures (Fahrenheit) in the state of Connecticut were 48, 48.2, 48, 48.7, according to the National Climatic Center Data web page. We store the state temperatures in `CT`, and compare the local New Haven temperatures with the state averages. For example, one can compute the annual differences in mean temperatures. Here `CT` and `temps` are both vectors of length four and the subtraction operation is applied element by element. The result is the vector of four differences.

```
> CT = c(48, 48.2, 48, 48.7)
> temps - CT
[1] 3.9 3.6 3.9 4.3
```

The four values in the result are differences in mean temperatures for 1968 through 1971. It appears that on average New Haven enjoyed slightly warmer temperatures than the state of Connecticut in this period.

Example 1.2 (President's heights). An article in Wikipedia [54] reports data on the heights of Presidents of the United States and the heights of their opponents in the presidential election. It has been observed [53, 48] that the taller presidential candidate typically wins the election. In this example, we explore the data corresponding to the elections in the television era. In Table 1.1 are the heights of the presidents and their opponents in the U.S. presidential elections of 1948 through 2008, extracted from the Wikipedia article.

Table 1.1 Height of the election winner in the Electoral College and height of the main opponent in the U.S. Presidential elections of 1948 through 2008.

Year	Winner	Height		Opponent	Height	
2008	Barack Obama	6 ft 1 in	185 cm	John McCain	5 ft 9 in	175 cm
2004	George W. Bush	5 ft 11.5 in	182 cm	John Kerry	6 ft 4 in	193 cm
2000	George W. Bush	5 ft 11.5 in	182 cm	Al Gore	6 ft 1 in	185 cm
1996	Bill Clinton	6 ft 2 in	188 cm	Bob Dole	6 ft 1.5 in	187 cm
1992	Bill Clinton	6 ft 2 in	188 cm	George H.W. Bush	6 ft 2 in	188 cm
1988	George H.W. Bush	6 ft 2 in	188 cm	Michael Dukakis	5 ft 8 in	173 cm
1984	Ronald Reagan	6 ft 1 in	185 cm	Walter Mondale	5 ft 11 in	180 cm
1980	Ronald Reagan	6 ft 1 in	185 cm	Jimmy Carter	5 ft 9.5 in	177 cm
1976	Jimmy Carter	5 ft 9.5 in	177 cm	Gerald Ford	6 ft 0 in	183 cm
1972	Richard Nixon	5 ft 11.5 in	182 cm	George McGovern	6 ft 1 in	185 cm
1968	Richard Nixon	5 ft 11.5 in	182 cm	Hubert Humphrey	5 ft 11 in	180 cm
1964	Lyndon B. Johnson	6 ft 4 in	193 cm	Barry Goldwater	5 ft 11 in	180 cm
1960	John F. Kennedy	6 ft 0 in	183 cm	Richard Nixon	5 ft 11.5 in	182 cm
1956	Dwight D. Eisenhower	5 ft 10.5 in	179 cm	Adlai Stevenson	5 ft 10 in	178 cm
1952	Dwight D. Eisenhower	5 ft 10.5 in	179 cm	Adlai Stevenson	5 ft 10 in	178 cm
1948	Harry S. Truman	5 ft 9 in	175 cm	Thomas Dewey	5 ft 8 in	173 cm

Section 1.5 illustrates several methods for importing data from a file. In this example we enter the data interactively as follows. The continuation character + indicates that the R command is continued.

```
> winner = c(185, 182, 182, 188, 188, 188, 185, 185, 177,
+   182, 182, 193, 183, 179, 179, 175)
> opponent = c(175, 193, 185, 187, 188, 173, 180, 177, 183,
+   185, 180, 180, 182, 178, 178, 173)
```

(Another method for entering data interactively is to use the `scan` function. See Example 3.1 on page 79.) Now the newly created objects `winner` and `opponent` are each vectors of length 16.

```
> length(winner)
[1] 16
```

The year of the election is a regular sequence, which we can generate using the sequence function `seq`. Our first data value corresponds to year 2008, so the sequence can be created by

```
> year = seq(from=2008, to=1948, by=-4)
```

or equivalently by

```
> year = seq(2008, 1948, -4)
```

According to the Washington Post blog [53], Wikipedia misstates "Bill Clinton's height, which was measured during official medical exams at 6 foot-2-1/2, making him just a tad taller than George H.W. Bush." We can correct the height measurement for Bill Clinton by assigning a height of 189 cm to the fourth and fifth entries of the vector `winner`.

```
> winner[4] = 189
> winner[5] = 189
```

The sequence operator `:` allows us to perform this operation in one step:

```
> winner[4:5] = 189
```

The revised values of `winner` are

```
> winner
 [1] 185 182 182 189 189 188 185 185 177 182 182 193 183 179 179 175
```

Are presidents taller than average adult males? According to the National Center for Health Statistics, in 2005 the average height for an adult male in the United States is 5 feet 9.2 inches or 175.768 cm. The sample mean is computed by the `mean` function.

```
> mean(winner)
[1] 183.4375
```

Interestingly, the opponents also tend to be taller than average.

```
> mean(opponent)
[1] 181.0625
```

Next, we use vectorized operations to compute the differences in the height of the winner and the main opponent, and store the result in `difference`.

```
> difference = winner - opponent
```

An easy way to display our data is as a data frame:

```
> data.frame(year, winner, opponent, difference)
```

The result is displayed in Table 1.2. Data frames are discussed in detail in Section 1.4.

We see that most, but not all, of the differences in height are positive, indicating that the taller candidate won the election. Another approach to determining whether the taller candidate won is to compare the heights with the logical operator `>`. Like the basic arithmetic operations, this operation is vectorized. The result will be a vector of logical values (`TRUE`/`FALSE`) having the same length as the two vectors being compared.

Table 1.2 Data for Example 1.2.

```
> data.frame(year, winner, opponent, difference)
   year winner opponent difference
1  2008    185      175         10
2  2004    182      193        -11
3  2000    182      185         -3
4  1996    189      187          2
5  1992    189      188          1
6  1988    188      173         15
7  1984    185      180          5
8  1980    185      177          8
9  1976    177      183         -6
10 1972    182      185         -3
11 1968    182      180          2
12 1964    193      180         13
13 1960    183      182          1
14 1956    179      178          1
15 1952    179      178          1
16 1948    175      173          2
```

```
> taller.won = winner > opponent
> taller.won
 [1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[10] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

On the second line, the prefix [10] indicates that the output continues with the tenth element of the vector.

The `table` function summarizes discrete data such as the result in the vector `taller.won`.

```
> table(taller.won)
taller.won
FALSE  TRUE
    4    12
```

We can use the result of `table` to display percentages if we divide the result by 16 and multiply that result by 100.

```
> table(taller.won) / 16 * 100
taller.won
FALSE  TRUE
   25    75
```

Thus, in the last 16 elections, the odds in favor of the taller candidate winning the election are 3 to 1.

Several types of graphs of this data may be interesting to help visualize any pattern. For example, we could display a barplot of differences using the `barplot` function. For the plot we use the `rev` function to reverse the order of the differences so that the election year is increasing from left to right. We also provide a descriptive label for both axes.

```
> barplot(rev(difference), xlab="Election years 1948 to 2008",
+   ylab="Height difference in cm")
```

The barplot of differences in heights is shown in Figure 1.1.

It would also be interesting to display a scatterplot of the data. A scatterplot of loser's heights vs winner's height for election years 1798 through 2004 appears in the Wikipedia article [54]. A simple version of the scatterplot (not shown here) can be obtained in R by

```
> plot(winner, opponent)
```

Chapter 4 "Presentation Graphics" illustrates many options for creating a custom graphic such as the scatterplot from the Wikipedia article.

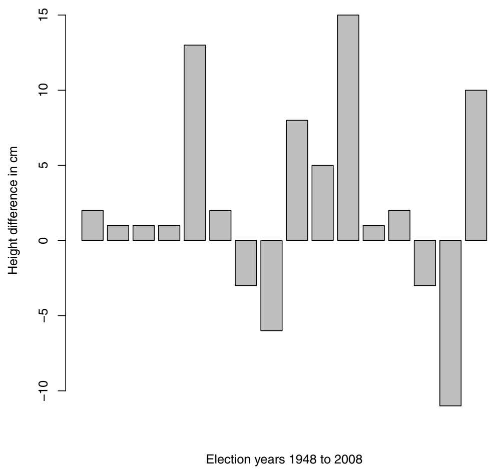

Fig. 1.1 Barplot of the difference in height of the election winner in the Electoral College over the height of the main opponent in the U.S. Presidential elections. Height differences in centimeters for election years 1948 through 2008 are shown from left to right. The electoral vote determines the outcome of the election. In 12 out of these 16 elections, the taller candidate won the electoral vote. In 2000, the taller candidate (Al Gore) did not win the electoral vote, but received more popular votes.

Example 1.3 (horsekicks). This data set appears in several books; see e.g. Larsen and Marx [30, p. 287]. In the late 19th century, Prussian officers collected data on deaths of soldiers in 10 calvary corps recording fatalities due to horsekicks over a 20 year period. The 200 values are summarized in Table 1.3.

To enter this data, we use the *combine* function `c`.

Table 1.3 Fatalities due to horsekick for Prussian calvary in Example 1.3

Number of deaths, k	Number of corps-years in which k fatalities occurred
0	109
1	65
2	22
3	3
4	1
	200

```
> k = c(0, 1, 2, 3, 4)
> x = c(109, 65, 22, 3, 1)
```

To display a bar plot of the frequencies, we use the `barplot` function. The function `barplot(x)` produces a barplot like Figure 1.2, but without the labels below the bars. The argument `names.arg` is optional; it assigns labels to display below the bars. Figure 1.2 is obtained by.

```
> barplot(x, names.arg=k)
```

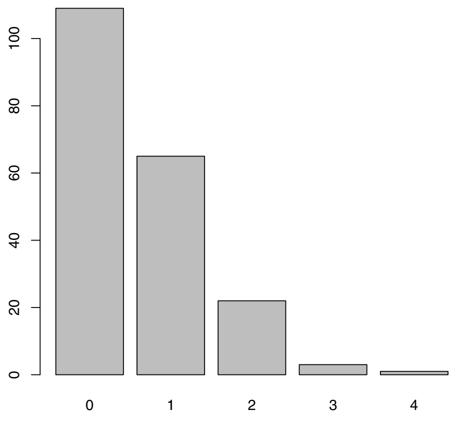

Fig. 1.2 Frequency distribution for Prussian horsekick data in Example 1.3.

The relative frequency distribution of the observed data in `x` is easily computed using vectorized arithmetic in R. For example, the sample proportion of 1's is $65/200 = 0.545$. The expression `x/sum(x)` divides every element of the vector x by the sum of the vector (200). The result is a vector the same length as x containing the sample proportions of the death counts 0 to 4.

```
> p = x / sum(x)
> p
[1] 0.545 0.325 0.110 0.015 0.005
```

The center of this distribution can be estimated by its sample mean, which is

$$\begin{aligned}\frac{1}{200}\sum_{i=1}^{200} x_i &= \frac{109(0)+65(1)+22(2)+3(3)+1(4)}{200}.\\ &= 0.545(0)+0.325(1)+0.110(2)+0.015(3)+0.005(4).\end{aligned}$$

The last line is simply the sum of p*k, because R computes this product element by element ("vectorized"). Now we can write the sample mean formula as the sum of the vector p*k. The value of the sample mean is then assigned to r.

```
> r = sum(p * k)
> r
[1] 0.61
```

Similarly, one can compute an estimate of the variance. Apply the computing formula for variance of a sample $y_1, \dots, y_n$:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \overline{y})^2.$$

Here the sample mean is the value r computed above and

$$s^2 = \frac{1}{n-1}\left\{109(0-r)^2 + 65(1-r)^2 + 22(2-r)^2 + 3(3-r)^2 + 1(4-r)^2\right\},$$

so the expression inside the braces can be coded as x*(k-r)^2. The sample variance v is:

```
> v = sum(x * (k - r)^2) / 199
> v
[1] 0.6109548
```

Among the counting distributions that might fit this data (binomial, geometric, negative binomial, Poisson, etc.) the Poisson is the one that has equal mean and variance. The sample mean 0.61 and sample variance 0.611 are almost equal, which suggests fitting a Poisson distribution to the data. The Poisson model has probability mass function

$$f(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k \geq 0, \tag{1.1}$$

where $\lambda = \sum_{k=0}^{\infty} kf(k)$ is the mean of the distribution. The sample mean 0.61 is our estimate of the population mean λ. Substituting the sample mean for λ in the density (1.1), the corresponding Poisson probabilities are

```
> f = r^k * exp(- r) / factorial(k)
> f
[1] 0.5433509 0.3314440 0.1010904 0.0205551 0.0031346
```

R has probability functions for many distributions, including Poisson. The R density functions begin with "d" and the Poisson density function is `dpois`. The probabilities above can also be computed as

```
> f = dpois(k, r)
> f
[1] 0.5433509 0.3314440 0.1010904 0.0205551 0.0031346
```

R_x 1.2 *R provides functions for the density, cumulative distribution function (CDF), percentiles, and for generating random variates for many commonly applied distributions. For the Poisson distribution these functions are `dpois`, `ppois`, `qpois`, and `rpois`, respectively. For the normal distribution these functions are `dnorm`, `pnorm`, `qnorm`, and `rnorm`.*

How well does the Poisson model fit the horsekick data? In a sample of size 200, the expected counts are $200f(k)$. Truncating the fraction using `floor` we have

```
>  floor(200*f)  #expected counts
[1] 108  66  20   4   0
>  x             #observed counts
[1] 109  65  22   3   1
```

for $k = 0, 1, 2, 3, 4$, respectively. The expected and observed counts are in close agreement, so the Poisson model appears to be a good one for this data.

One can alternately compare the Poisson probabilities (stored in vector `f`) with the sample proportions (stored in vector `p`). To summarize our comparison of the probabilities in a matrix we can use `rbind` or `cbind`. Both functions bind vectors together to form matrices; with `rbind` the vectors become rows, and with `cbind` the vectors become columns. Here we use `cbind` to construct a matrix with columns `k`, `p`, and `f`.

```
> cbind(k, p, f)
     k     p         f
[1,] 0 0.545 0.5433509
[2,] 1 0.325 0.3314440
[3,] 2 0.110 0.1010904
[4,] 3 0.015 0.0205551
[5,] 4 0.005 0.0031346
```

It appears that the observed proportions `p` are close to the Poisson(0.61) probabilities in `f`.

1.1.3 R Scripts

Example 1.3 contains several lines of code that would be tedious to retype if one wants to continue the data analysis. If the commands are placed in a file, called an R script, then the commands can be run using `source` or copy-paste. Using the `source` function causes R to accept input from the named source, such as a file.

Open a new R script for editing. In the R GUI users can open a new script window through the *File* menu. Type the following lines of "horsekicks.R" (below) in the script. It is a good idea to insert a few comments. Comments begin with a # symbol.

Using the `source` function, auto-printing of expressions does not happen. We added `print` statements to the script so that the values of objects will be printed.

`horsekicks.R`

```
# Prussian horsekick data
k = c(0, 1, 2, 3, 4)
x = c(109, 65, 22, 3, 1)
p = x / sum(x)        #relative frequencies
print(p)

r = sum(k * p)   #mean
v = sum(x * (k - r)^2) / 199  #variance
print(r)
print(v)
f = dpois(k, r)
print(cbind(k, p, f))
```

At this point it is convenient to create a working directory for the R scripts and data files that will be used in this book. To display the current working directory, type `getwd()`. For example, one may create a directory at the root, say `/Rx`. Then change the working directory through the File menu or by the function `setwd`, substituting the path to your working directory in the quotation marks below. On our system this has the following effect.

```
> getwd()
[1] "C:/R/R-2.13.0/bin/i386"
> setwd("c:/Rx")
> getwd()
[1] "c:/Rx"
```

Save the script as "horsekicks.R" in your working directory. Now the file can be sourced by the command

```
  source("horsekicks.R")
```

and all of the commands in the file will be executed.

R_x 1.3 *Unlike* Matlab *.m files, an R script can contain any number of functions and commands.* Matlab *users may be familiar with defining a function* *by writing an .m file, where each .m file is limited to exactly one function. Function syntax is covered in Section 1.2.*

R_x 1.4 *Here are a few helpful shortcuts for running part of a script.*

- *Select lines and click the button 'Run line or selection' on the toolbar.*
- *Copy the lines, and then paste the lines at the command prompt.*
- *(For Windows users:) To execute one or more lines of the file in the R GUI editor, select the lines and type Ctrl-R.*
- *(For Macintosh users:) One can execute lines of a file by selecting the lines and typing Command-Return.*

Example 1.4 (Simulated horsekick data). For comparison with Example 1.3, in this example we use the random Poisson generator `rpois` to simulate 200 random observations from a Poisson($\lambda = 0.61$) distribution. We then compute the relative frequency distribution for this sample. Because these are randomly generated counts, each time the code below is executed we obtain a different sample and therefore the results of readers will vary slightly from what follows.

```
> y = rpois(200, lambda=.61)
> kicks = table(y)   #table of sample frequencies
> kicks
y
  0   1   2   3
105  67  26   2
> kicks / 200        #sample proportions
y
    0     1     2     3
0.525 0.335 0.130 0.010
```

Comparing this data with the theoretical Poisson frequencies:

```
> Theoretical = dpois(0:3, lambda=.61)
> Sample = kicks / 200
> cbind(Theoretical, Sample)
  Theoretical Sample
0  0.54335087  0.525
1  0.33144403  0.335
2  0.10109043  0.130
3  0.02055505  0.010
```

The computation of mean and variance is simpler here than in Example 1.3 because we have the raw, ungrouped data in the vector `y`.

```
> mean(y)
[1] 0.625
> var(y)
[1] 0.5571608
```

It is interesting that the observed Prussian horsekicks data seems to fit the Poisson model better than our simulated Poisson($\lambda = 0.61$) sample.

1.1.4 The R Help System

The R Graphical User Interface has a Help menu to find and display online documentation for R objects, methods, data sets, and functions. Through the Help menu one can find several manuals in PDF form, an html help page, and help search utilities. The help search utility functions are also available at the command line, using the functions `help` and `help.search`, and the corresponding shortcuts `?` and `??`. These functions are described below.

- `help("keyword")` displays help for “keyword”.
- `help.search("keyword")` searches for all objects containing “keyword”.

The quotes are usually optional in `help`, but would be required for special characters such as in `help("[")`. Quotes are required for `help.search`. When searching for help topics, keep in mind that R is case-sensitive: for example, `t` and `T` are different objects.

One or two question marks in front of a search term also search for help topics.

- `?keyword` (short for `help(keyword)`).
- `??keyword` (short for `help.search("keyword")`).

Try entering the following commands to see their effect.

```
?barplot        #searches for barplot topic
??plot          #anything containing "plot"

help(dpois)         #search for "dpois"
help.search("test") #anything containing "test"
```

The last command above displays a list including a large number of statistical tests implemented in the R.

One of the features of R online help is that most of the keywords documented include examples appearing at the end of the page. Users can try one or more of the examples by selecting the code and then copy-paste to the console. R also provides a function `example` that runs all of the examples if any exist for the keyword. To see the examples for the function `mean`, type `example(mean)`. The examples are then executed and displayed at the console with a special prompt symbol (`mean>`) that is specific to the keyword.

```
> example(mean)

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

```
mean> mean(USArrests, trim = 0.2)
  Murder  Assault UrbanPop     Rape 
    7.42   167.60    66.20    20.16 
>
```

For many of the graphics functions, the documentation includes interesting examples. Try `example(curve)` for an overview of what the `curve` function can do. The system will prompt the user for input as it displays each graph.

A glossary of R functions is available online in "Appendix D: Function and Variable Index" of the manual "Introduction to R" [49], and the "R Reference Manual" [41] has a comprehensive index by function and concept. These manuals are included with the R distribution, and also available online on the R project home page[1] at the line "Manuals" under "Documentation".

1.2 Functions

The R language allows for modular programming using functions. R users interact with the software primarily through functions. We have seen several examples of functions above. In this section, we discuss how to create user-defined functions.

The syntax of a function is

```
f = function(x, ...) {
  }
```

or

```
f <- function(x, ...) {
  }
```

where `f` is the name of the function, `x` is the name of the first argument (there can be several arguments), and `...` indicates possible additional arguments. Functions can be defined with no arguments, also. The curly brackets enclose the body of the function. The return value of a function is the value of the last expression evaluated.

Example 1.5 (function definition). R has a function `var` that computes the unbiased estimate of variance, usually denoted by s^2. Occasionally, one requires the maximum likelihood estimator (MLE) of variance,

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 = \frac{n-1}{n}\, s^2.$$

A function to compute $\hat{\sigma}^2$ can be created as follows.

[1] www.r-project.org

```
var.n = function(x) {
  v = var(x)
  n = NROW(x)
  v * (n - 1) / n
  }
```

The `NROW` function computes the number of observations in `x`. The value `v * (n-1)/n` evaluated on the last line is returned. Note: it would also be correct (but unnecessary) to replace the last line of the function `var.n` with

```
    return(v * (n - 1) / n)
```

Before this user-defined function can be used, one must input the code so that the function, in this case `var.n`, is an object in the R workspace. Normally, one places functions in a script file and uses the `source` function (or copy and paste to the command line) to submit them. Here is an example that computes s^2 and $\hat{\sigma}^2$ for the temperature data of Example 1.1.

```
> temps = c(51.9, 51.8, 51.9, 53)
> var(temps)
[1] 0.3233333
> var.n(temps)
[1] 0.2425
```

Example 1.6 (functions as arguments). Many of the available R functions require functions as arguments. An example is the `integrate` function, which implements numerical integration; one must supply the integrand as an argument. Suppose that we need to compute the beta function, which is defined as

$$B(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}\,dx,$$

for constants $a > 0$ and $b > 0$. First we write a function that returns the integrand evaluated at a given point x. The additional arguments a and b specify the exponents.

```
f = function(x, a=1, b=1)
    x^(a-1) * (1-x)^(b-1)
```

The curly brackets are not needed here because there is only one line in the function body. Also, we defined default values $a = 1$ and $b = 1$, so that if a or b are not specified, the default values will be used. The function can be used to evaluate the integrand along a sequence of x values.

```
> x = seq(0, 1, .2)  #sequence from 0 to 1 with steps of .2
> f(x, a=2, b=2)
[1] 0.00 0.16 0.24 0.24 0.16 0.00
```

This *vectorized* behavior is necessary for the function argument of the `integrate` function; the function that evaluates the integrand must accept a vector as its first argument and return a vector of the same length.

Now the numerical integration result for $a = b = 2$ can be obtained by