A Step-by-Step R Tutorial

An introduction into R applications and programming

Niël J le Roux; Sugnet Lubbe

bookboon
The eBook company

Niël J le Roux & Sugnet Lubbe

A Step-by-Step R Tutorial:

An introduction into R applications and programming

A Step-by-Step R Tutorial: An introduction into R applications and programming
2nd edition

ISBN 978-87-403-2144-9

Contents

	Preface	**11**
	Acknowledgement	**13**
1	**Introducing the R system**	**14**
1.1	Introduction	14
1.2	Downloading the R system	14
1.3	A quick sample R session	15
1.4	R: An interpretive computer language	17
1.5	A closer look at the R console	19
1.6	More R basics	23
1.7	Regular expressions in R: The basics	26
1.8	From single instructions to sets of instructions: Introducing R functions	28
1.9	R within the Windows environment	33
1.10	Working with RStudio	36
1.11	Using R Commander	38
1.12	Activating an R project	39

1.13	A note on computations by a computer	40
1.14	Built-in data sets in R	40
1.15	The use of `.First()` and `.Last()`	40
1.16	Options	42
1.17	R output (text and graphics) to Microsoft Word	42
1.18	Creating PDF and HTML documents from R output: R package `knitr`	43
1.19	Command line editing	48
2	**Managing objects**	**49**
2.1	Creating separate workspaces for each project	49
2.2	Instructions and objects in R	50
2.3	How R finds data	56
2.4	The organization of data (data structures)	61
2.5	Time series	62
2.6	The functions `as.xxx()` and `is.xxx()`	62
2.7	Simple manipulations; numbers and vectors	62
2.8	Objects, their modes and attributes	63
2.9	Representation of objects	64
2.10	Exercise	66

3	**R operators and functions**	**68**
3.1	Arithmetic operators	68
3.2	Logical operators	71
3.3	The operators `<-`, `<<-` and ~	73
3.4	Operator precedence	73
3.5	Introduction to functions in R	74
3.6	Some mathematical functions	75
3.7	Differentiation and integration	85
3.8	Exercise	85
4	**Introducing traditional R graphics**	**87**
4.1	General	87
4.2	High-level plotting instructions	88
4.3	Interactive communication with graphs	92
4.4	3D graphics: Package `rgl`	93
4.5	Exercise	94

5	**Subscripting**	**97**
5.1	Subscripting with vectors	97
5.2	Subscripting with matrices	98
5.3	Extracting elements of lists	104
5.4	Extracting elements from dataframes	107
5.5	Combining vectors, matrices, lists and dataframes	109
5.6	Rearranging the elements in a matrix	110
5.7	Exercise	110
6	**Revision tasks**	**111**
6.1	Guidelines for problem solving by writing R code	111
6.2	Exercise	111
7	**Writing functions in R**	**117**
7.1	General	117
7.2	Writing a new function	121
7.3	Checking for object name clashes	121
7.4	Returning multiple values	122
7.5	Local variables and evaluation environments	122
7.6	Cleaning up	123

7.7	Variable number of arguments: `argument ...`	124
7.8	Retrieving names of arguments: Functions `deparse()` and `substitute()`	125
7.9	Operators	127
7.10	Replacement functions	127
7.11	Default values and lazy evaluation	129
7.12	The dynamic loading of external routines	129
8	**Vectorized programming and mapping functions**	**132**
8.1	Mapping functions to a matrix	132
8.2	Mapping functions to vectors, dataframes and lists	134
8.3	The functions: `mapply()`, `rapply()` and `Vectorize()`	134
8.4	The mapping function `tapply()` for grouped data	135
8.5	The control of execution flow statement `if-else` and the control functions `ifelse()` and `switch()`	135
8.6	Loops in R	138
8.7	The execution time of R tasks	140
8.8	The calling of functions with argument lists	141
8.9	Evaluating R strings as commands	142
8.10	Object-oriented programming in R	142
8.11	Recursion	145

8.12 Environments in R 146
8.13 "Computing on the language" 146
8.14 Writing user friendly functions in R: The function `menu()` 146
8.15 Exercise 148
8.16 The function `on.exit()` 148
8.17 Error tracing 149
8.18 Error handling: The function `try()` 150

9 Reading data files into R, formatting and printing 153
9.1 Reading Microsoft Excel files into R 153
9.2 Reading other data files into R 153
9.3 Sending output to a file 154
9.4 Writing R objects for transport 154
9.5 The use of the file .Rhistory and the function `history()` 154
9.6 Command re-editing 154
9.7 Customized printing 155
9.8 Formatting numbers 156
9.9 Printing tables 156
9.10 Communicating with the operating system 158
9.11 Exercise 158

10 R graphics: Round II 159
10.1 Graphics parameters 159
10.2 Layout of graphics 160
10.3 Low-level plotting commands 161
10.4 Using the plotting commands 163
10.5 Exact distances in graphics 174
10.6 Multiple graphics windows in R 175
10.7 More complex layouts 175
10.8 Dynamic 3D graphics in R 177
10.9 Animation 177
10.10 Exercise 178

11 Statistical modelling with R 179
11.1 Introduction 179
11.2 Data for statistical models 180
11.3 Expressing a statistical model in R 180
11.4 Common arguments to R modelling functions 181
11.5 Using the statistical modelling objects 182

11.6	Usage of the function `with()`	183
11.7	Linear regression and anova	183
11.8	The function `glm()`	187
11.9	The function `gam()`	187
11.10	The function `loess()`	189
11.11	The function `rpart()`	189
11.12	Nonlinear regression and the function `nls()`	189
11.13	Normal quantile plot	191
11.14	A coplot with two conditioning variables	192
11.15	Regression diagnostics	193
11.16	Experimental design	194
11.17	Consider the following data	194
12	**Analysis of Variance and Covariance with R**	**195**
12.1	One-way ANOVA models	195
12.2	Two-way ANOVA models: Main effects and interaction effects	196
12.3	One-way ANCOVA models	199
12.4	Maize example	204
12.5	Heart rate example	205
13	**Introduction to Optimization**	**206**
13.1	The bisection method for solving $f(x) = 0$	206
13.2	The Newton-Raphson method	209
13.3	The R functions `optim()` and `constrOptim()`	212
13.4	Packages `quadprog`, `lpSolve` and `Rsolnp` for constrained optimization	213
	References and Further Reading	**222**
	Endnotes	**223**
	Index	**224**

Preface

The R system is an open-source software project for analyzing data and constructing graphics. It provides a general computer language for performing tasks like organizing data, statistical analyses, simulation studies, model fitting, building of complex graphics and many more.

Central to the R system is the high-level R computer language. Its roots date back to the birth of the computer language S on May 5, 1976 at Bell Labs, Murray Hill, New Jersey (Chambers, 2008). In its early days S underwent several revisions and extensions mainly for implementation on the UNIX operating system. Eventually an enhanced version of S was licensed under the name S-PLUS and became available for the Windows operating system under the name S-PLUS for Windows. The earlier versions of R adhered to the principles of functional programming and with the release of version S3 in the middle eighties its building blocks were dynamically generated, self-describing objects. The publication *The New S Language* (Becker, Chambers and Wilks, 1988) provides a detailed description of S3. The next major development of S was the release of *Statistical Models in S* (Chambers and Hastie (Eds), 1993) which involved the merging of the functional style of S with object-oriented programming concepts of classes and methods. However, S3 has only limited formal support for classes and methods. The introduction of S4 objects (Chambers, 1998) introduced a new class and method system but retains S3 compatibility. In the meantime several versions of S-PLUS based upon S3 at first and later on S4 were released in the commercial market.

The R language itself was introduced in a paper published by Ross Ihaka and Robert Gentleman of Auckland, New Zealand in 1996 (Ihaka and Gentleman, 1996). This proposal was to a large extent compatible with S but included features from the Lisp/Scheme family of languages. An important aspect of R was its availability as an open-source system.

Both R and S-PLUS can be considered to be clones of the same underlying S. That means that if you are able to program in the one you can quite easily program in the other but be warned: there are also fundamental differences between the two systems.

In the first decade of the twenty-first century interest in R has exceeded all possible expectations. Apart from a well maintained core system with new releases every few months there are currently literally thousands of researchers contributing add-on packages on cutting-edge developments in statistics and data analysis.

This book is a tutorial with a twofold aim; learning the basics of the R system and how to program efficiently in R. It is the result of an introductory course in S-PLUS taught at the University of Stellenbosch since 1995. The initial course was based on the book *An Introduction to S and S-Plus* (Spector, 1994). Since 2002 increasingly more emphasis was put on R to such an extent that it is currently almost exclusively devoted to R. This change necessitated the preparation of class notes for a ten day (eight hours a day) tutorial course in R. The result is *A Step-by-Step R Tutorial: An introduction into R applications and programming*.

Acknowledgement

We would like to acknowledge with enormous appreciation the impact of Spector, P. (1994): An Introduction to S and S-Plus (Pacific Grove, CA: Duxbury Press) on the programming courses the authors have been directed since 1995 and specifically on A Step-by-Step R Tutorial. In the early days we used Spector as our handbook but the advent of R and the specific needs at our own institutions necessitated important changes in our approach during the last decade leading to A Step-by-Step R Tutorial.

Niel le Roux and Sugnet Lubbe 2015.

1 Introducing the R system

1.1 Introduction

This chapter introduces the R system to the new R user. The Windows operating system is emphasized but most of the material covered also applies to other operating systems after allowing for the requirements of the particular operating system in use. Users with some experience with R should quickly glance through this chapter making sure they have mastered all topics covered here before proceeding with the main tutorial starting with Chapter 2.

In the computer age statistics has become inseparable from being able to write computer programs. Therefore, let us start with a reminder of the Fundamental Goal of S:
Conversion of an idea into useful software

The challenge is to pursue this goal keeping in mind the Mission of R (Chambers, 2008):
…to enable the best and most thorough exploration of data possible

and its Prime Directive (Chambers, 2008):
…places and obligation on all creators of software to program in such a way that the computations can be understood and trusted

1.2 Downloading the R system

Website for downloading R: http://www.R-project.org
To download R to your own computer: Navigate to …./*bin/windows/base* and save the file *R-3.1.x.-win.exe* on your computer. Click this file to start the installation procedure and select the defaults unless you have a good reason not to do so. By default, if you are installing on a 32 bit machine, the following R icon will be created on your desktop:

If you are installing on a 64 bit machine the following two icons will be created on your desktop:

In the latter case you can run 32 bit or 64 bit applications by clicking the appropriate icon.

The core R system that is installed includes several ***packages***. Apart from these installed packages several thousands of dedicated ***contributed packages*** are available to be downloaded by users in need of any of them.

1.3 A quick sample R session

Click the R icon created on your desktop to open the ***Commands Window*** or ***Console***. Notice the R prompt > waiting for some instruction from the user.

a) At the R prompt > enter 5 – 8. We will follow the following convention to write instructions as this:

```
> 5 - 8 ↵
```

b) Repeat (a) but enter only 5 – and see what happens:

```
> 5 - ↵
> + ↵
> +
```

The above + is the secondary R prompt. It indicates that an instruction is unfinished. Either respond by completing the instruction or press the Esc button to start all over again from the primary prompt.

c) Enter

```
> xx <- 1:10 ↵
```

This instruction creates an R object with name (or label) `xx` containing the vector (1, 2, 3, 4, 5, 6, 7, 8, 10).

d) Enter

```
> yy <- rnorm(n = 20, mean = 50, sd = 15) ↵
```

This instruction creates an R object with name *yy* containing a random sample of 20 values from a normal distribution with a mean of 50 and a standard deviation of 15.

e) Enter

```
> xx ↵
[1]  1  2  3  4  5  6  7  8  9 10
```

The above example shows that ***when the name of an R object is entered at the prompt, R will respond by displaying the contents of the object.***

f) Obtain a representation of the contents of the object `yy` created in (d).

g) A program in R is called a ***function***. Any function in R is also an R ***object*** and therefore has a name (or label). It follows from (e) that if the name of a function is entered at the prompt, R will respond by displaying the contents of the function.
How then can an R function be executed i.e. how can an R function be called? Apart from its name an R function has a list of arguments enclosed within parentheses. An R function is called by entering at the prompt its name followed by a list of arguments enclosed within parentheses. As an example let us calculate the mean of the object `yy` created above by calling the function mean:

```
> mean(yy)
```

Note that the prompt appear followed by the mean of object `yy`.

h) Objects created during an R session are stored in a database *.RData* in the working directory or workspace. A listing of all the objects in a database can be obtained by calling the functions `ls()` or `objects()`. Now, first enter at the R prompt the instruction `objects` (or `ls`) and then the instruction `objects()` (or `ls()`). Explain what has happened.

i) Objects can be removed by the following instruction: `rm(name1, name2, … )`

j) Apart from the ***console*** there are several other types of windows available in R e.g. graphs are displayed in graph windows. To illustrate, enter the following instructions at the R prompt in the console or commands window:

```
> gr.data <- rnorm(1000)
> hist(gr.data)
```

These instructions have resulted in the opening of a graph window containing the required histogram and the user can switch from the console to the graph window and back again to the console.

k) The R session can be terminated by closing the window or entering `q()` at the R prompt. Either way the user is prompted to save the workspace. If the user chooses not to save all objects created during the session are lost.

1.4 R: An interpretive computer language

Essentially, in an interpretive language instructions are given one by one. Each instruction is then evaluated or interpreted in turn by an internal program called an ***interpreter*** or ***evaluator*** and some immediate action is taken. For example, the instruction given in §1.3(a) is evaluated by the R evaluator resulting in the answer –3 being returned. On the other hand in §1.3(b) the evaluator found the instruction to be incomplete and therefore asked for more information.

An advantage of an interpretive language is that intermediate results can be obtained quickly without having first to wait for a complete program to finish as is the case with a compiler language. In the latter case a complete program is translated (or compiled) by a program called a compiler. The compiled program can then be converted to a standalone application that can be called by other programs to perform a complete task. In general compiler languages handle computer memory relatively more efficiently and calculations are executed more speedily.

Communication with the R evaluator takes place through a set of instructions called ***escape sequences***. These escape sequences take the form of a backslash preceding a character. Examples of such escape sequences are:

`\n`	new line
`\r`	carriage return
`\t`	go to next tab stop
`\b`	backspace
`\a`	bell
`\f`	form feed
`\v`	vertical tab

A consequence of the above role of the backslash in R is that a single backslash in a filename will not be properly recognized. Therefore, when referring in R to the following file path `"c:\My Documents\myFile.txt"` all backslashes must be entered as double backslashes i.e. `"c:\\My Documents\\myFile.txt"`.

Exercise 1.4.1

The `cat()` function can be used to write a text message to the console. Initialize a new R session and investigate the results of the following R instructions:

```
> cat("aaa bbb")
> cat("aaa bbb \n")
> cat("aaa \n bbb \n")
> cat("aaa \nbbb \n")
> cat("aaa \t\t bbb \n")
> cat("aaa\b\b\bbbb \n")
> cat("aaa \n\a bbb \a\n")
> cat("1\a\n"); cat("2\a\n")
```

What is the purpose of the semi-colon in the line above?
Could you distinguish the two soundings of the bell?

```
> cat("1\a\n"); Sys.sleep(2); cat("2\a\n")
```

Could you now distinguish the two soundings of the bell?
What is the purpose of the `Sys.sleep()` instruction?

Exercise 1.4.2

Write R code to achieve the following output:

```
My name is:
Bell sounds once.
Your name appears on a new line.
Two distinct sounds of the bell are heard and
Thank you  is visible on a new line.
The cursor appears on a new line.
```

1.5 A closer look at the R console

An example of the R console (also called the R commands window) is given in Figure 1.5.1. The R console consists of

a) A top line with the dropdown menus *File*, *Edit*, *View*, *Misc*, *Packages*, *Windows*, *Help*

b) A toolbar of buttons allowing several shortcuts. Different buttons are available depending on the active window. When the console itself is the active window there are buttons for *Open script*, *Load workspace*, *Save workspace*, *Copy*, *Paste*, *Copy and paste*, *Stop current computation*, *Print*

c) The console itself where commands are given interactively at the R prompt

d) The statusbar at the bottom.

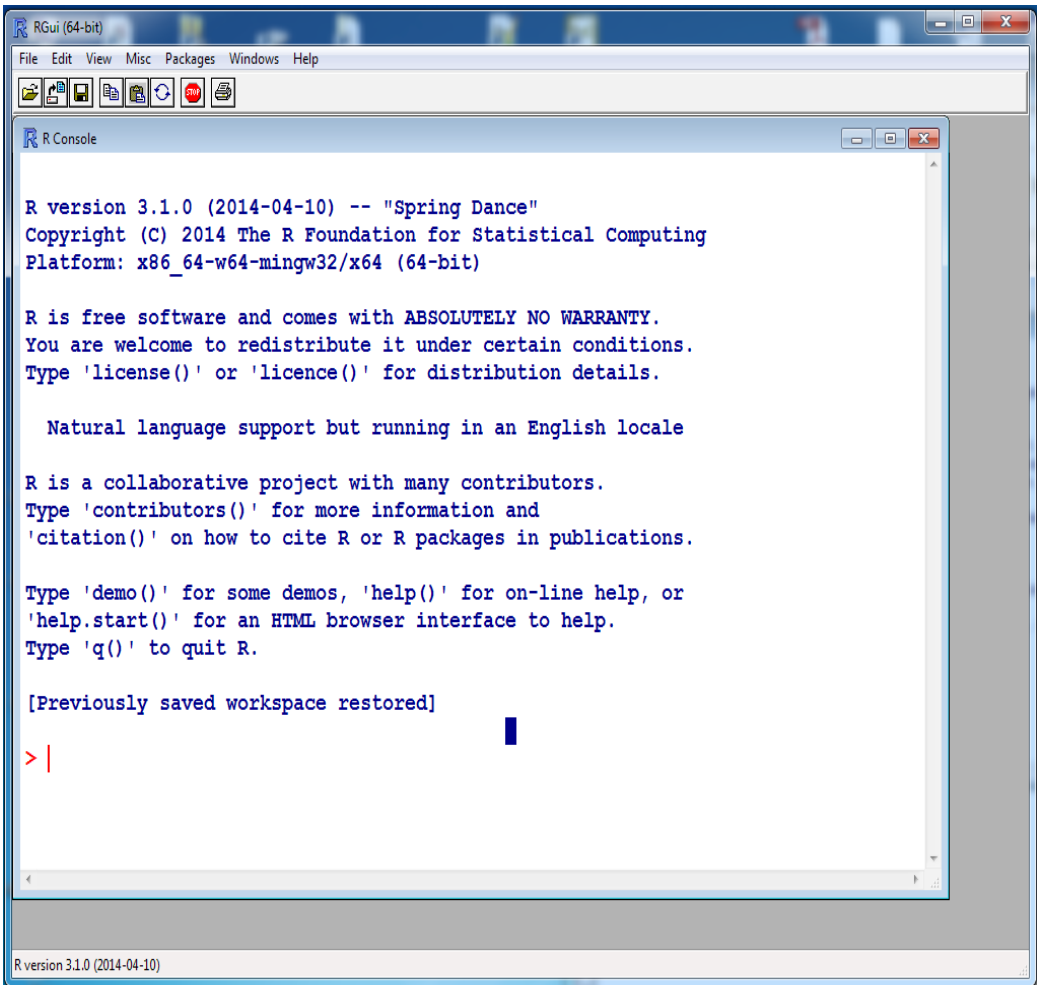
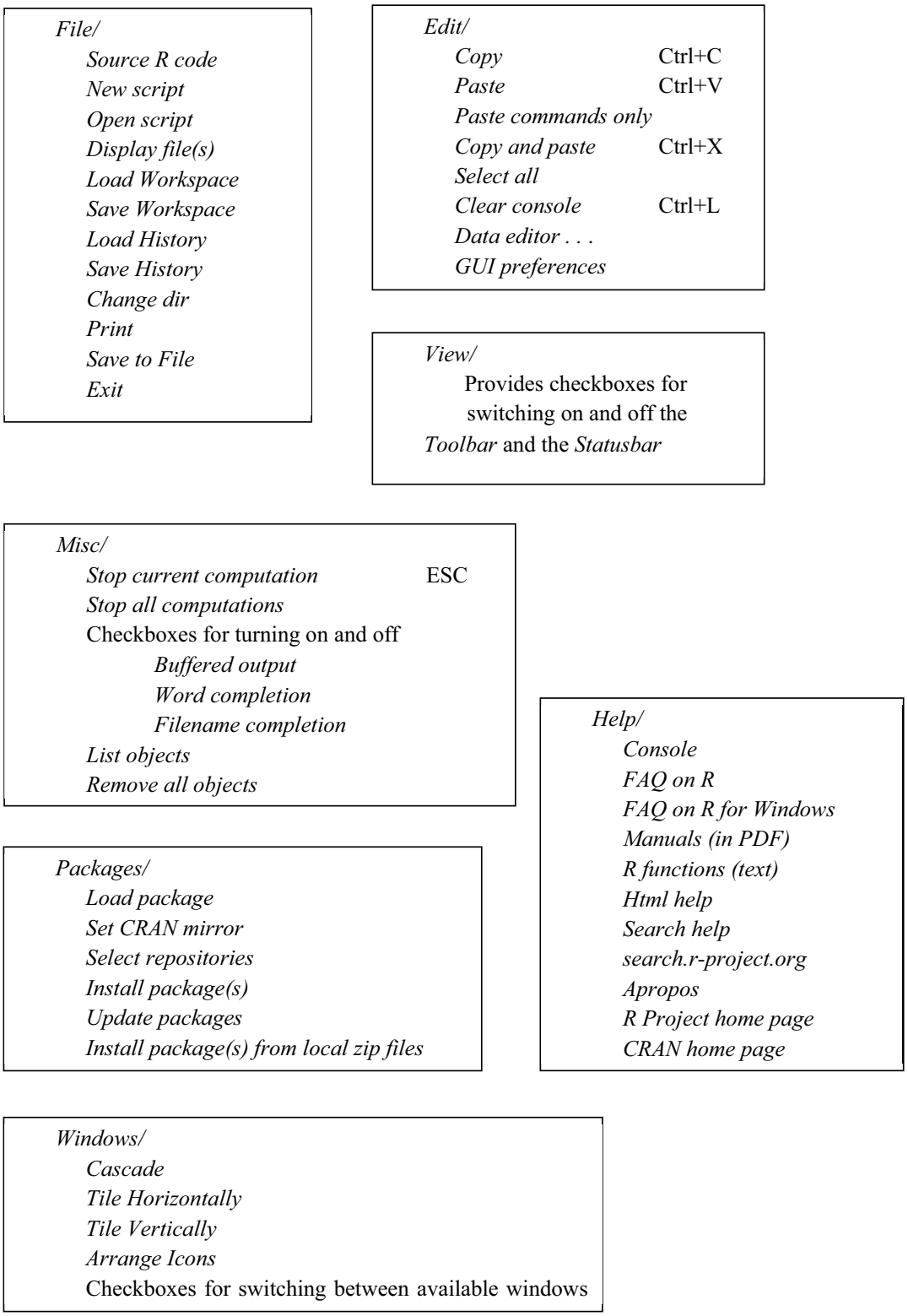


Figure 1.5.1: The R console (R commands window).

The dropdown menus provide the following choices and facilities:



Familiarize yourself with the usage of the available tools from the R console. Several of these tools will be discussed in more detail in later chapters.

Exercise 1.5

Initialize an R session

a) Use *Edit; GUI preferences* to change the console font to be Arial, 12pt, red, bold, on a yellow background.

b) Use *Help; Manuals (in PDF)* to find out what manuals are available for studying the R system in detail.

c) Use *Help; R Project home page* to find out what information is available on the R Project home page.

d) Use *Help; CRAN home page* to find out what contributed packages are available for download.

e) Use

```
> ?mean ↲
```

to obtain help on the usage of the R function `mean()`.

f) Find out what is the difference between the instructions

```
> ?mean ↲
```

and

```
> ??mean ↲
```

g) What help is available via the instruction

```
> help.start() ↲
```

h) Use

```
> ?help.search() ↲
```

to find out how to obtain help using the R function `help.search(xx)`

Note: For help on an operator or reserved word quotes are needed e.g.

```
> ?matrix ↲
```

but

```
> ?"?" ↲
```

or

```
> ?"for" ↲
```

1.6 More R basics

a) R as an ***interactive*** language allows for fast acquisition of results.

b) R is a ***functional*** language in two important senses: In a more technical sense it means the R model of computation relies more on ***function evaluation*** than by procedural computations and changes of state. The second sense refers to the way how users communicate to R namely almost entirely through ***function calls***.

c) R as an ***object-oriented*** language refers in a technical sense to the S4 type of objects with their associated classes and methods as mentioned in the *Preface*. In a less technical sense it means that everything in R is an object

d) R objects will be studied in detail in later chapters. What is important for now, is the following:

- Everything in R is an object.
- There are different types of objects e.g. function objects, data objects, graphics objects, character objects, numeric objects.
- Usually objects are stored in the working directory called the ***Global environment***; recognized by R under the name `.GlobalEnv` and available in the file system under the name `.RData`

- Objects are created from the console by ***assignment*** through the instruction

  ```
  > name <- object
  ```

  or

  ```
  > object -> name
  ```

- In R names are ***case sensitive*** i.e. *peter* and *Peter* are two different objects.

- Objects created by assignment during an R session are stored permanently in the Global environment (working directory) unless the user chooses not to save when terminating an R session.

- Care must be exercised when creating a new object by assignment: if an object with the name `my.object` already exists in the Global environment and a new object is created by assigning it to the name `my.object` then the old `my.object` is over-written and it is replaced by the new object ***without any warning***.

- Remember the way the R evaluator operates: if an object name is given at the R prompt the R evaluator responds by displaying the content of the object. Review the difference between the instructions

  ```
  > q ↵
  ```

  and

  ```
  > q() ↵
  ```

e) The symbol # marks a comment. Everything following a # on a line is ignored by the R evaluator. Check for example what is the result of the instruction

```
> 5+8 # +12 ↵
```

f) Usage of the symbols <−, = and ==. The symbol <− is used for assigning the object on its right-hand side to a name (label) on its left-hand side; the equality sign = is used for specifying the arguments of functions while the double equality symbol == is used for comparison purposes. In earlier versions of R these rules were strictly applied by the R evaluator. However, in recent versions of R the evaluator allows the equality sign also in the case for assigning an object to a name. We believe that reserving the equality sign only for argument specifications in functions leads to more clarity when writing complex functions and therefore we discourage its usage for creating objects by assignment. In this book creating objects by assignment will be exclusively carried out with the assignment symbol <−.

g) The symbol −> assigns the object on its left-hand side to the name (label) on its right-hand side.

h) Working with packages: The core installation includes several packages. To see them issue the command `search()` from the R prompt in the console. Notice that the first object in the search list is `.GlobalEnv`. This is followed by other objects. Packages are recognized by the string *package* followed by a colon and the name of the package. In order for a package to be used the following steps must be followed: if the package has been ***installed*** previously it needs only to be ***loaded*** into the search path using the command `library(packagename)` from the R prompt. This will load the package by default in the second position on the search path. If the package has not been installed previously it must first be installed. This is most easily done using the top menu *Packages*.

i) More on the help (?) facility: Table 1.6.1 contains details about help available for some special keywords.

Help query	Explanation
`?Arithmetic`	Unary and binary operators to perform arithmetic on numeric and complex vectors
`?Comparison`	Binary operators for comparison of values in vectors
`?Control`	The basic constructs for control of the flow in R instructions
`?dotsMethods`	The use of the special operator…
`?Extract`	Operators to extract or replace parts of vectors, matrices, arrays and lists
`?Logic`	Logical operators for operating on logical and numeric vectors
`?.Machine`	Information on the variable .Machine holding information on the numerical characteristics of the machine R is running on
`?NumericConstants`	How R parses numeric constants including Inf, NaN, NA
`?options`	Allow the user to set and examine a variety of global *options* which affect the way in which R computes and displays its results
`?Paren`	Parentheses and braces in R
`?Quotes`	Single and double quotation marks. Back quote (backtick) and backslash for starting an escape sequence
`?Reserved`	Description of reserved words in R
`?Special`	Special mathematical functions related to the beta and gamma functions including permutations and combinations
`?Syntax`	Outlines R syntax and gives the precedence of operators

Table 1.6.1: Some useful keywords available for help queries.

1.7 Regular expressions in R: The basics

It follows from §1.6 (d) that care must be taken when objects are assigned to names. Furthermore, the Global environment or any other R database may easily contain hundreds of objects. Therefore, a frequent task is to search for patterns in the names of objects e.g. searching for all object names starting with "*Figure.*" or ending in "*.dat*". The R function `objects()` or `ls()` has arguments `pos` and `pattern` for specifying the position of a database to search and a pattern of characters appearing in a name (or string), respectively. The `pattern` argument can be given any ***regular expression***. Regular expressions provide a method of expressing patterns in character values and are used to perform various tasks in R. Here we are only considering the task of extracting certain specified objects in a database using the `pattern` argument of `objects()` or `ls()`.

The syntax of regular expressions follows different rules than the syntax of ordinary R instructions. Moreover its syntax differs depending on the particular implementation a program uses. By default, R uses a set of regular expressions to those used by UNIX utilities but function arguments are available for changing the default e.g. by setting argument `perl=TRUE`.

Regular expressions consist of three components: ***single characters***; ***character classes*** and ***modifiers*** operating on single characters and character classes.

Character classes are formed by using square brackets surrounding a set of characters to be matched e.g. `[abc123]; [a-z]; [a-zA-Z]; [0-9a-z]`. Note the usage of the dash to indicate a range of values.

The modifiers operating on characters or character classes are summarized in Table 1.7.1.

Modifier	Operation
^	Expression anchors at beginning of target string
$	Expression anchors at end of target string
.	Any single character except newline is matched
|	Alternative patterns are separated
( )	Patterns are grouped together
*	Zero or more occurrences of preceding entity are matched
?	Zero or one occurrences of preceding entity are matched
+	One or more occurrences of preceding entity are matched
{*n*}	Exactly *n* occurrences of preceding entity are matched
{*n*,}	At least *n* occurrences of preceding entity are matched
{*n*, *m*}	At least *n* and at most *m* occurrences of preceding entity are matched

Table 1.7.1: Modifiers for regular expressions.

Because of their role as modifiers or in forming character classes the following characters must be preceded by a backslash when their literal meaning is needed:

[ ] { } ( ) ^ $. | * + \

Note that in R this means that whenever one of the above characters needs to be escaped in a regular expression it must be preceded by double backslashes. Table 1.7.2 contains some examples of regular expressions.

Regular expression	Meaning
`"[a-z][a-z][0-9]"`	Matches a string consisting of two lower case letters followed by a digit
`"[a-z][a-z][0-9]$"`	Matches a string ending in two lower case letters followed by a digit
`"^[a-zA-Z]+\\."`	Matches a string beginning with any number of lower or upper case letters followed by a period
`"(ab){2}(34){2}$"`	Matches a string ending in `abab3434`

Table 1.7.2: Examples of regular expressions.

Exercise 1.7

Initialize an R session

a) Attach the `MASS` package in the second (the default) position on the search path by issuing the command

```
> library(MASS) ↲
```

b) Get a listing of all the objects in package `MASS` by requesting

```
> objects(pos=2) ↲
```

c) Explain the difference between `objects(pos=2, pat=".")` and `objects(pos=2, pat="\\.")`.

d) Obtain a listing of all objects with names starting with three letters followed by a digit.

e) Obtain a listing of all objects with names ending with three letters followed by a digit.

f) Obtain a listing of all objects with names ending in a period followed by exactly three or four letters.

1.8 From single instructions to sets of instructions: Introducing R functions

Consider the following problem: the R data set `sleep` contains the extra hours of sleep of 20 patients after a drug treatment. Suppose this data set can be considered a sample from a normal population. A 95% confidence interval is required for the mean extra hours of sleep. It is known that the confidence interval is given by $\left[\bar{x} - \left(\frac{s}{\sqrt{n}}\right) t_{n-1;0.025};\ \bar{x} + \left(\frac{s}{\sqrt{n}}\right) t_{n-1;0.025}\right]$. This problem can be solved by entering the following instructions one by one:

```
> sleep.data <- sleep[ ,1]
> sleep.mean <- mean(sleep.data)
> sleep.sd <- sqrt(var(sleep.data))
> t.perc <- qt(0.975,19)
> left.boundary <- sleep.mean-(sleep.sd/sqrt(20))*t.perc
> right.boundary <- sleep.mean+(sleep.sd/sqrt(20))*t.perc
```

In situations like the above, the problem can be addressed using a ***script file*** or writing a ***function***. We are going to introduce two methods for writing functions in R:

1. using a script file and
2. using the function `fix()`.

Both the above two methods make use of a ***text editor***. The built-in ***R text editor*** can be used when using script files but in the windows environment *notepad* or preferably *notepad++* (available for free from www.notepad-plus-plus.org/download/) or *Tinn-R* is preferred.

The following instruction is necessary for changing the default editor to be used with `fix()`:

```
> options(editor = "notepad") or
> options(editor = "full path to the relevant exe file")
```

1.8.1 Writing an R function using a script file

a) From the R top menu select *File; New script*. A script window will open (see Figure 1.8.1) with a simultaneous change in the menu bar. Note the name of the editor appearing in the script window.

b) Type the instructions in the script window as shown in the figure below.

c) Select all the typed text and run the script by clicking the run icon (or *Ctrl+R*).

d) Note what is shown in the R console window.

e) Script files are ordinary text files. They can be saved, edited and opened using any text editor.

f) By convention R script files have the extension *XX.r*.

g) Next, change the spelling of the last right.boundary to Right.boundary. Select all the text and run the script. Check the output appearing on the console. Explain.

h) Script windows can also be used for creating an R function – see Figure 1.8.2 on the next page.

i) Create an R function by changing the text in the script as shown in the Figure 1.8.2.

j) Select the text and notice what happens in the R commands window (the console).

k) Give the instruction `objects()` at the R prompt. What has happened?

l) If you want to create and run the function `my.func1` in a script window then add the instruction `my.func1(x=sleep)` as the last line in the script window. Now, select only this line and run it. Check the R console.

m) What will happen if a syntax error is made in the script window? Change the code `my.func1` in Figure 1.8.2 to `my.func2` and deliberately delete the last closing parenthesis in the last line of the script window. Select all code in the script and run it. Check the R console. Discuss.

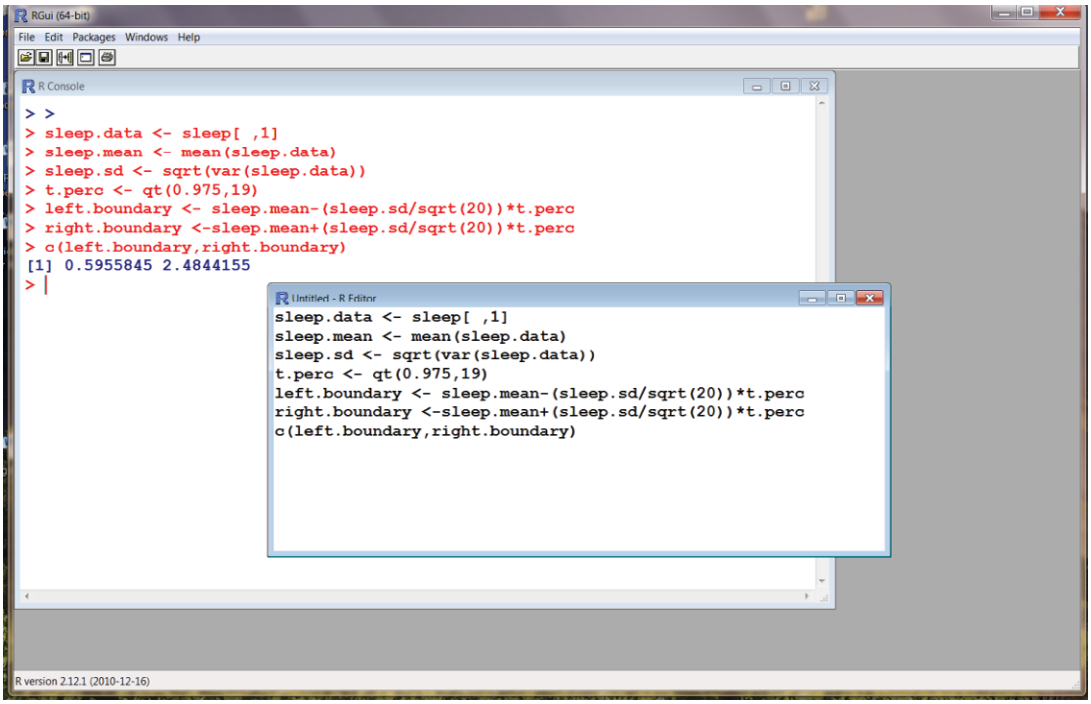


Figure 1.8.1: The R console with an opened script file window.

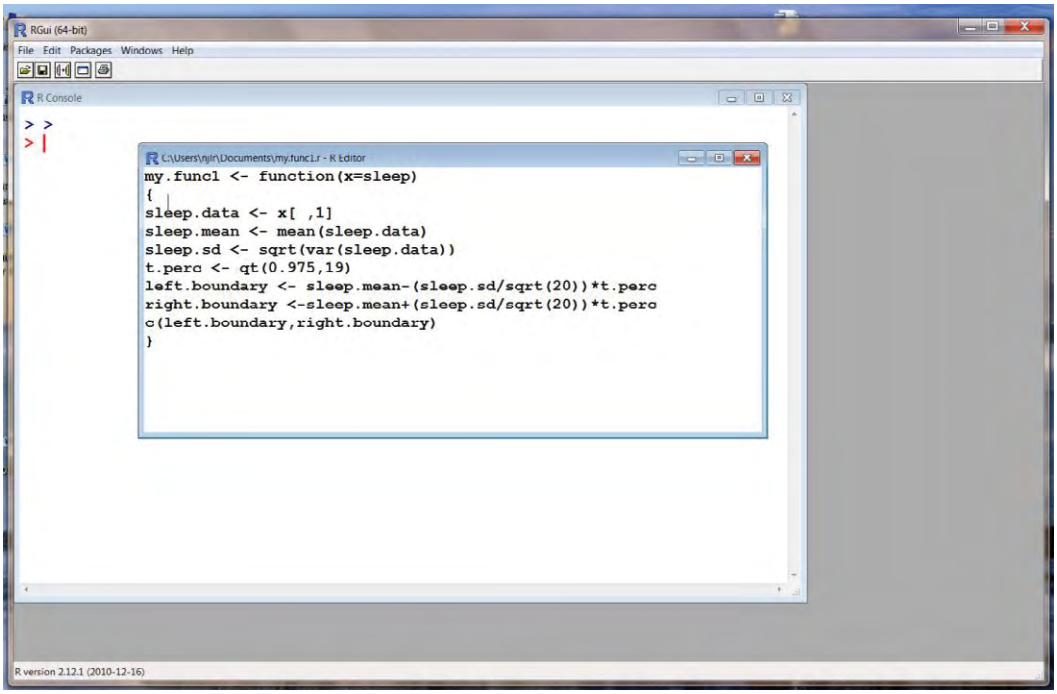


Figure 1.8.2: R console with script file window showing the function `my.func1()`.

1.8.2 Writing an R function using fix()

a) Make sure the default text editor is notepad or notepad++ (or any other text editor of your choice). This can be done using the following instruction from the R prompt:

```
> options (editor="notepad")
```

or by providing the full path to your preferred text editor, i.e. something similar to

```
> options (editor="C:\\Program files\\Notepad++\\Notepad++")
```

b) Enter fix (my.func3) at the R prompt. A text editor will open. See Figure 1.8.3 below. Type the instructions as shown in the editor window in Figure 1.8.3. Close the window. Check what happens in the R console.

c) What will happen if a syntax mistake is made when using fix? To see: First copy `my.func3` to `my.func4`. Then, at the R prompt type `fix (my.func4)`. Make a deliberate syntax error, e.g. delete the last closing brace. Close the text editor window. What happens in the console? What is to be done to correct the mistake?

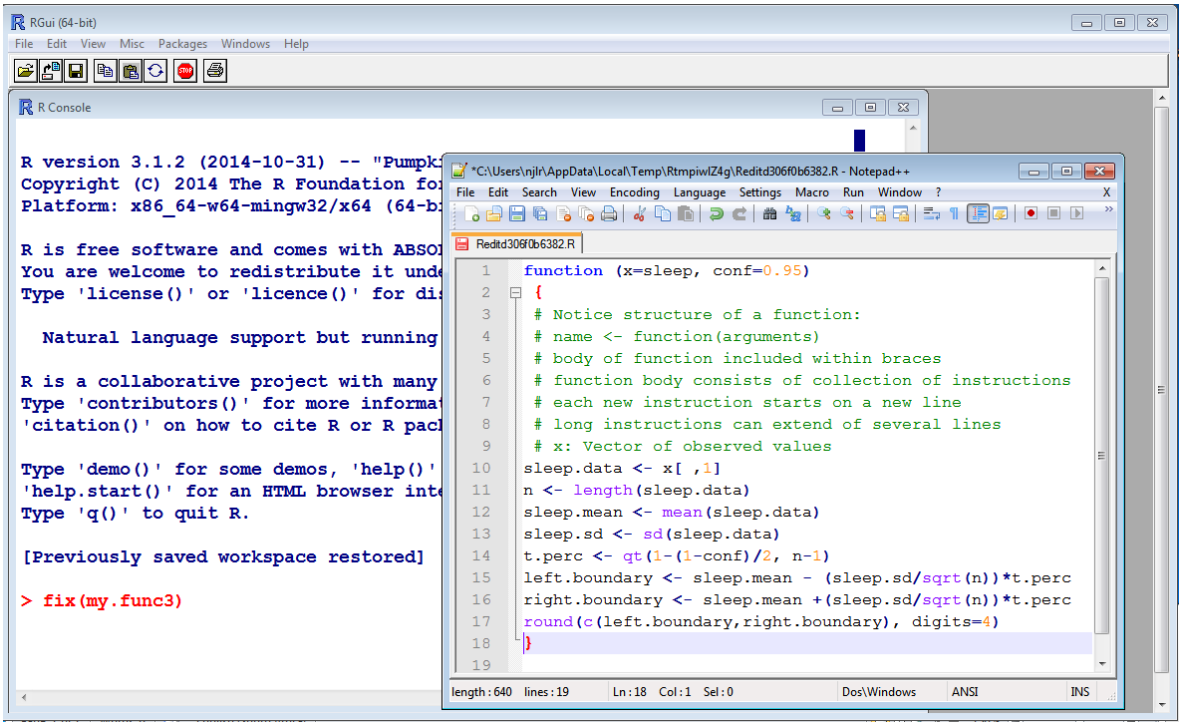


Figure 1.8.3: R console together with a Notepad++ window. Notice some of the editing facilities available in the Notepad++ window.

d) Carefully study the message in the R console when a syntax error occurred in a function created by fix():

```
> Error in edit(name, file, title, editor) :
unexpected 'yyy' occurred on line xx
    use a command like
    x <- edit()
    to recover
```

e) Explain why is the following a correct way to respond to the above message from the R evaluator:

```
> my.func4 <- edit() ↵
```

WARNING

Before writing a function for solving any problem: make sure the problem is understood exactly; make 100% sure the relevant statistical theory is understood correctly. Failure to do so is careless and dangerous!

1.9 R within the Windows environment

The different windows in R are the Data window, Script window, Graph window and Menus and Dialog windows. The current workspace in R is `.GlobalEnv`. The function `getwd()` is used to obtain the path to the working directory's `.RData` and `.Rhistory` *Note*: In order to see the the files *.RData* and *.Rhistory* being displayed as such, it may be necessary to turn off the option "Hide extensions for known file types" in *Control Panel*; *Folder Options*; *View*.

a) Ordering of projects in different workspaces in R
   It is important to make provision for different workspaces associated with different projects. Below is an example of how different projects can be ordered in different working directories. Each directory has the same icon, but has a different name. By left-clicking on the correct icon, the respective project's folder is accessed. Therefore, all the objects that belong to a specific project are kept together.

b) How can a group of directories similar to that in Figure 1.9.1 be obtained?
   i. Right-click on any open space on the Desktop and select *New*; *Folder*.

   ii. Right-click on the *New folder* and select *Properties*; *Customize*; *Change icon*; *Browse*.

   iii. Navigate to …\Program files\R\R-3.1.x\bin\x64\Rgui.exe; *OK* to assign the characteristic **R** icon of R to the group of directories.

   iv. Change the name of *New folder* to, for example, *R-projects*. Keep on desktop or copy to any location you prefer e.g. D:\R-projects.

   v. Ensure you have a folder to hold the `.RData` and `.Rhistory` files for all your different R projects e.g. the folder D:\R_Workspaces.

   vi. Left-click on the R-projects folder (open the directory group). Now right-click on any open space of the opened folder and select: *New*; *Shortcut*; *Browse* and navigate as above to …\Program files\R\R-3.1.x\bin\x64\Rgui.exe; *OK*. Choose *Finish* after providing a suitable descriptive name for the shortcut.

   vii. Next, right-click the newly created named shortcut in (vi) and select *Properties*. A properties window is opened as shown in Figure 1.9.1. Edit the *Start in* box to e.g. 'D:\R_Workspaces\BayesProject\' . Select *OK*.

viii. We have now created within the R-projects directory group a shortcut and if this shortcut is activated, an R-session is initialized with the corresponding `.RData` and `.Rhistory` files created in the chosen location 'D:\R_Workspaces\BayesProject\'

ix. As shown in Figure 1.9.1 the shortcut in (vii) can now be duplicated, appropriately named for each different R-project and the correct location for creating the necessary `.RData` and `.Rhistory` files provided.

x. Repeat step (ix) for every separate project (each with own path and name) that you want to put in the R-projects folder. Select any of these icons in order to begin a session with the respective project.

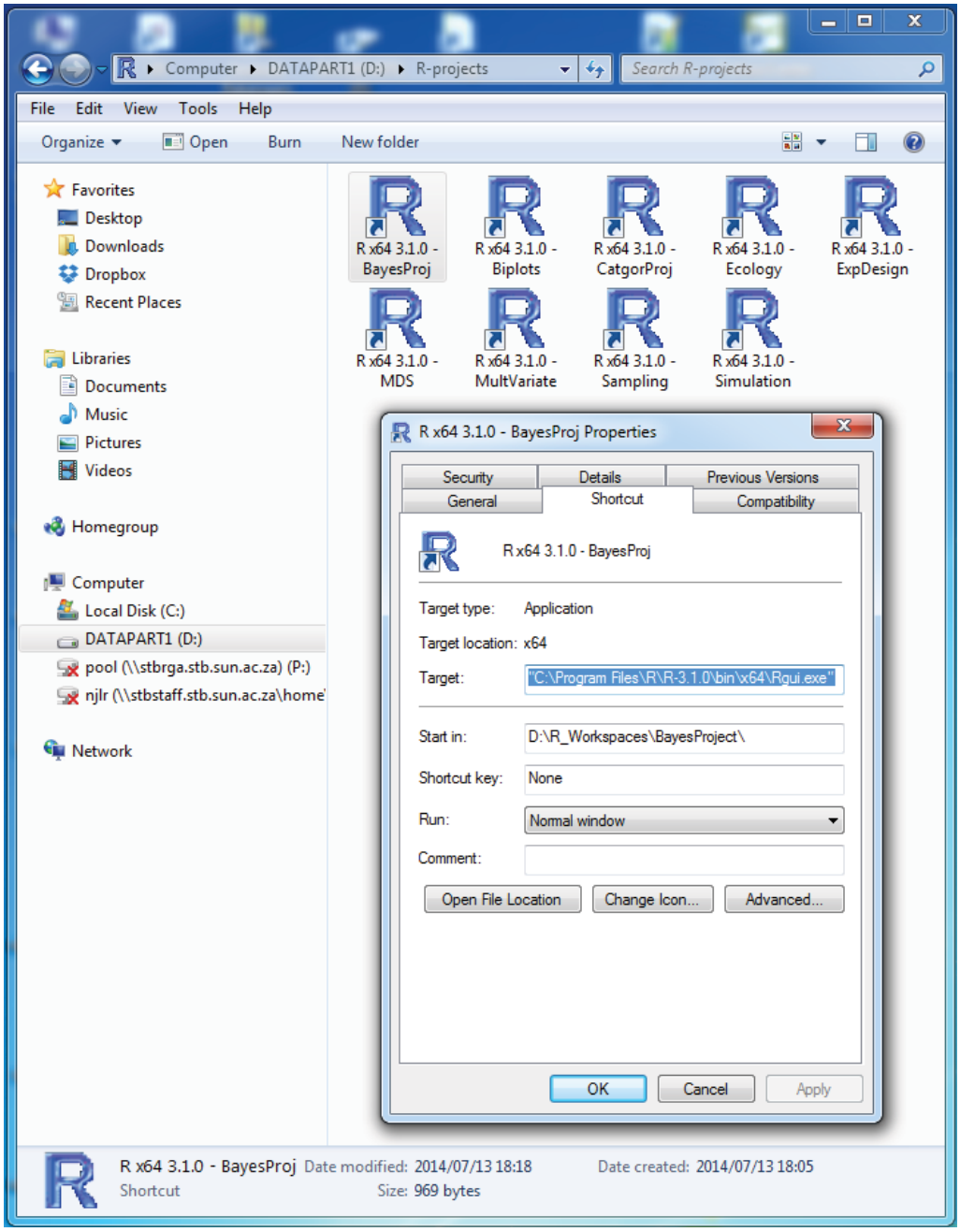


Figure 1.9.1: Several R working directories arranged in a Windows group.

c) Study how the following instructions work in R:

- *File*; *Load workspace*
- *File*; *Save workspace*
- *File*; *Change Dir*

d) An alternative method for allocating each R project its own working directory
   i. Create a folder for the project e.g. C:\MyProject
   ii. Click on any existing R icon to open an R session
   iii. In R, select from the top menu: *File*; *Save Workspace*
   iv. Navigate to C:\MyProject and select: *Save*
   v. Close R session
   vi. In Windows go to C:\MyProject and select *.RData*
   vii. It may be necessary to prepare the *.RData* file by (a) creating a suitable `.First` object and (b) by deleting all unnecessary objects e.g. by the instruction

```
> rm(list = objects()) (BE CAREFUL, Why?)
```

1.10 Working with RStudio

Many users of R prefer working with RStudio. RStudio is a free and open source integrated development environment for R which works with the standard version of R available from CRAN. It can be downloaded from the RStudio home page www.rstudio.com to be run from your desktop (Windows, Mac or Linux). Full details about the functionality of RStudio are available from its home page. Here, only a brief introduction to RStudio is given.

When RStudio is installed on your computer the following icon is created on the desktop:

Clicking the above icon open the RStudio development environment as shown in Figure 1.10.1. In order to open any R workspace with RStudio drag the corresponding `.RData` file to the above RStudio icon and drop it as soon as 'Open with RStudio' becomes visible.

The bottom left-hand panel is the familiar R console.

The bottom right-hand panel is used for:

a) a listing of the files in the folder where the workspace (`.RData`) for the active project is kept

b) a listing of all installed packages available to be attached to the search path as well as menus for installing and updating packages

c) the graph windows (if any)

d) the Help facilities.

The top left-hand panel can be used for creating and managing script files while the top right-hand panel provides information on the objects in the working directory as well as the history of previous commands given in the console.

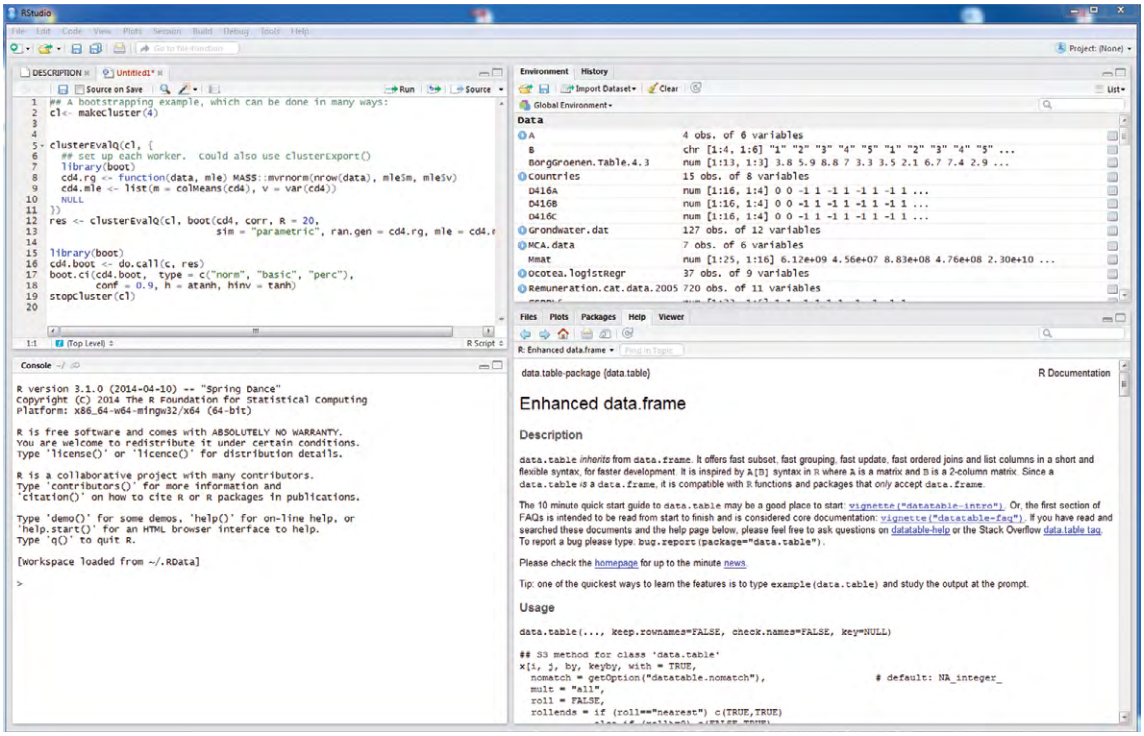


Figure 1.10.1: The RStudio development environment for R.

1.11 Using R Commander

The R Commander provides a graphical user interface (GUI) to many of the facilities available in the R system. Rather than focussing on programming, facilities in the form of drop down menus are provided to perform statistical analyses, data manipulation and creating reports. It is available in the R package `Rcmdr`. To open the R Commander initiate an R session and run the command

```
> library(Rcmdr) ↵
```

This opens the R Commander GUI as illustrated in Figure 1.11.1. For an introduction on getting started with the R Commander and some of its capabilities select *Help* in the top menu of R Commander and then *Introduction to the R Commander*.

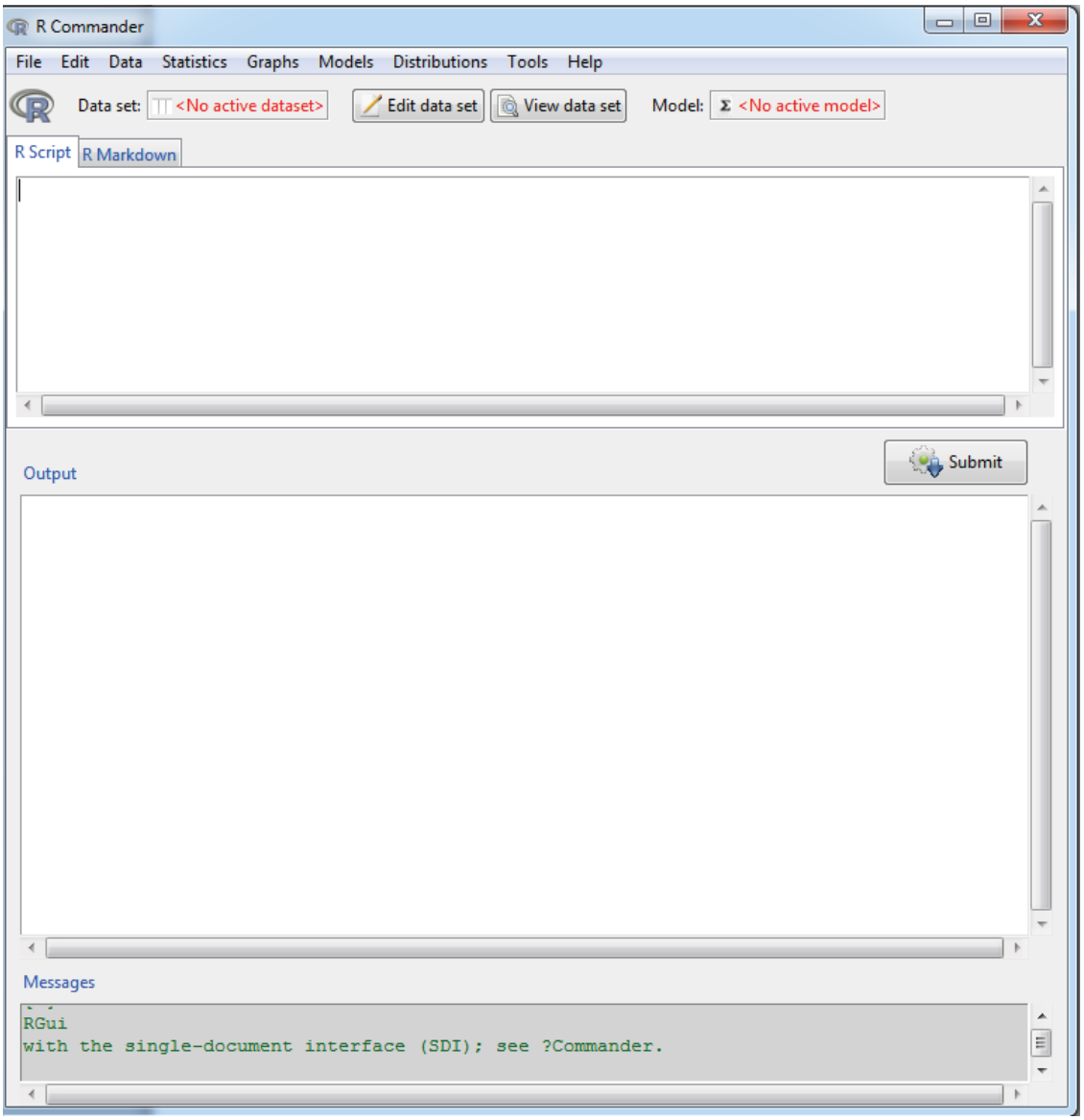


Figure 1.11.1: The R Commander graphical users interface.

1.12 Activating an R project

Since an R workspace can be saved as a *.RData* file in any chosen folder an R session can be initialized with any particular R project just by clicking the relevant *.RData* file. However, if RStudio has been installed, the project will be opened in an RStudio environment. If the user wants the project to be opened within R itself the following method provides an easy way for that: Drag the *.RData* file to the icon R itself creates on the desktop during its installation. Position the *.RData* file on the R icon until the message *Open with R for Windows GUI front end* becomes visible before dropping it onto the icon. This will open the required R session. Make sure that the *.RData* file is not dropped while the message *Copy to Desktop* is displayed. As indicated above this method can also be followed to open an RStudio session.

1.13 A note on computations by a computer

When writing R functions it is important to keep in mind that the way computations are performed by a computer are not always according to the rules of algebra. Two important occurrences are given below.

- In mathematics the following statement is wrong: `x = x + k` for `k ≠ 0` but in computer programming the statement `x = x + k` is legitimate and it means *x is replaced by* `x + k`.
- In general the treatment of integers and real numbers for which R uses floating point representation happens at a fundamental level over which R has no control. In general real numbers cannot necessarily be exactly represented in a computer – some can only be approximated. Furthermore, there are limitations to the minimum and maximum numbers that can be represented in a computer. This might lead to what is known as ***underflow*** or ***overflow***. A more detailed discussion appears in a later chapter.

Open an R session and issue the command
`> .Machine` ↲
for details about the numerical environment of your computer.

1.14 Built-in data sets in R

R contains several built-in data sets collected in the package `datasets`. This package is automatically attached to the search path. Type `?datasets` at the R prompt for details. Apart from these data sets several other data sets from other packages are also used in this book. Data sets available in certain files may be read into R using instructions as:

```
> my.data.object <- read.table(file = "Full path\\filename")
```

Note the use of the double \\.

1.15 The use of .First() and .Last()

The function `.First()` is executed at the beginning of every R session. Therefore

```
> .First <- function(){options(editor = "notepad")}
```

ensures that "Notepad" is the text editor during any subsequent session without having to specify

```
> options(editor = "notepad")↲
```

each time an R session is initialized.
Similar to `.First()` the function `.Last()` can be created for execution at the end of an R session.

1.15.1 Security: An example of the usage of .First()

The `.First` facility can be used to prevent access to a R working directory by setting a password protection. This can be done as follows:

Create a new workspace for running the example on security. In this workspace create the following R function

```
password <- function() # Note the structure of a function
 { cat("Please type in your UNQUOTED password. \n")
      password <- readline() # What is the usage of readline()?
      if (password != "PASSWORD") q(save="no")
                    # The meaning of != is "not equal to"
      else (cat("You can proceed \n"))
      invisible()
 }
```

Now create the function:

```
.First <- function()
      { # What must you be careful of?
          password()
       }
```

- Terminate your R session and open it again.
- Discuss the construction and usage of the above functions.
- Can you break the above security?
- Can you make changes to the above security to make it more safe?

1.16 Options

a) Study the result of the instruction `> options()` ↲ in R.

b) Study the R top menu choice *Edit; GUI preferences.*

1.17 R output (text and graphics) to Microsoft Word

R can be used together with MS Word (and Microsoft Powerpoint) to round off statistical analyses with a report. The basic principles are illustrated with the following example:

a) Initialize an R session and open an *XX.docx* file.

b) Attach the package MASS to the search path using the command

```
> library(MASS) ↲
```

c) Check the search path using the command

```
> search() ↲
```

d) Obtain the information about the `Cars93` data set in the `MASS` package using the command

```
> ?Cars93 ↲
```

e) Execute the following R instructions:

- `apply (Cars93[,4:8], 2, mean)`
- `hist (Cars93[,8])`

f) Interpret the results obtained in (e).

g) Select the applicable R text output and copy it to the *XX.docx* file.

h) Activate the *XX.docx* file and change the font of the copied R text output to

- Times New Roman
- Courier New

i) What very important difference is noticed? What is the reason for this?

j) Now activate the R graph window. Copy the graph by either of the following
   - *Ctrl+W* shortcut
   - *File; Copy to clipboard; as a Metafile*

k) Activate *XX.docx* and execute the following: *Edit; Paste special; Picture*

l) Select the graph in *XX.docx*. Resize the graph after ascertaining that the check box for ***Lock aspect ratio*** is checked. Study the different options to change the size, layout, colours and other characteristics of the graph.

m) Important: In order to correctly interpret many types of graphs in statistics it is essential that the correct aspect ratio is maintained in the graph. The procedure in (l) ensures that the aspect ratio of the original graph as created in R is maintained when resizing it in the Word document.

n) What is a wrong way of resizing a graph constructed in R in MS Word?

o) Remark: Powerpoint provides various facilities for fine-tuning graphs. (e.g., changing the colours and plotting characters).

1.18 Creating PDF and HTML documents from R output: R package `knitr`

The R package `knitr` is used to obtain reproducible results from R code in the form of PDF or HTML documents. We present here only the basic capabilities of the `knitr` package but the student is urged to experiment with this tool when processing individual assignments.

First we use a text editor to create in the same folder where the active R Workspace is located, the *.rnw* file *Example.1.18.1.rnw* given below:

```
\documentclass[12pt, a4paper]{article}
\usepackage{amsfonts, graphicx}
\begin{document}

\title{An Illustration of some Capabilities of \emph{knitr}}

\author{Niel le Roux and Sugnet Lubbe}

\maketitle
Code chunks in .Rnw files are delimited with $<<>>=$ at
the top where a chunk label and any chunk options can appear within the inner
$<>$ with an @ following. In-line R code chunks are indicated
with \verb+\+$Sexpr{}$ with the R code inside the curly braces.
```

\verb+ ##+

Here is an example containing several chunks of code. Note that in the first chunk R code is not shown due to the option $echo = FALSE$. In the remaining chunks R code is shown due to the option $echo = TRUE$.

```
<<data,echo=FALSE>>=
# Note R code not shown for this chunk
y<-1
# y
require(lattice)
set.seed(123)
x <- rnorm(1000, 20, 5)
@
```

We analyze data drawn from $\mathcal{N}(20 ,25)$. The mean is \Sexpr{round(mean(x), 3)}. Figure~\ref{fig:fig1} shows the distribution via a histogram and Figure~\ref{fig:fig2} shows it via a boxplot.

```
\begin{figure}[h!]
<<hist, echo= TRUE>>=
par(mar = c(1,4,1,4))
```

```
histogram(x)
@
\caption{\label{fig:fig1}Histogram}
\end{figure}
\begin{figure}[h!]
<<boxplot>>=
par(mar = c(0.5,4,1,4))
boxplot(x)
@
\caption{\label{fig:fig2}Boxplot}
\end{figure}

The first element of \texttt{x} is \Sexpr{x[1]}.
Note the usage of \verb+\texttt{x}+ above.
\begin{figure}
<<side.by.side.plots, fig.show='hold',fig.width=4, fig.height=4, out.width='.4\\linewidth'>>=
## two plots side by side (option fig.show='hold')
par(mar=c(4,4,.1,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,.7,0),tcl=-.3,las=1)
boxplot(x)
hist(x,main='')
@
\caption{\label{fig:fig3}Boxplot and Histogram side-by-side}
\end{figure}
\begin{table}
<<kable, results="asis">>=
n <- 100
x <- rnorm(n)
y <- 2*x + rnorm(n)
out <- lm(y ~ x)
summary(out)$coef
kable(summary(out)$coef, digits=2, format = 'latex')
@
\caption{Estimated regression coefficients}
\end{table}
\end{document}
```

Study the above file carefully. Notice that it consists of ordinary text, chunks of R code and LateX code. Next from R issue the following commands:

```
> library(knitr) ↲
```

```
> knit('Example.1.17.rnw')↲
```

This last instruction results in the file *Example.1.18.tex* to be created in the same folder where the current R workspace is located. Assuming that Windows users have MiKTeX (http://miktex.org/) or some other TeX interpreter installed the file *Example.1.18.pdf* can be built:

An Illustration of some Capabilities of *knitr*

Niel le Roux and Sugnet Gardner-Lubbe

December 3, 2014

Code chunks in .Rnw files are delimited with <<>>= at the top where a chunk label and any chunk options can appear within the inner <> with an @ following. In-line R code chunks are indicated with *Sexpr* with the R code inside the curly braces.

##

Here is an example containing several chunks of code. Note that in the first chunk R code is not shown due to the option $echo = FALSE$. In the remaining chunks R code is shown due to the option $echo = TRUE$.

```
## Loading required package:  lattice
```

We analyze data drawn from $\mathcal{N}(20, 25)$. The mean is 20.081. Figure 1 shows the distribution via a histogram and Figure 2 shows it via a boxplot.

The first element of `x` is 17.1976218. Note the usage of `\texttt{x}` above.

Figure 1.18.1: Output in the file *Example.1.18.pdf*

```
par(mar = c(1,4,1,4))
histogram(x)
```

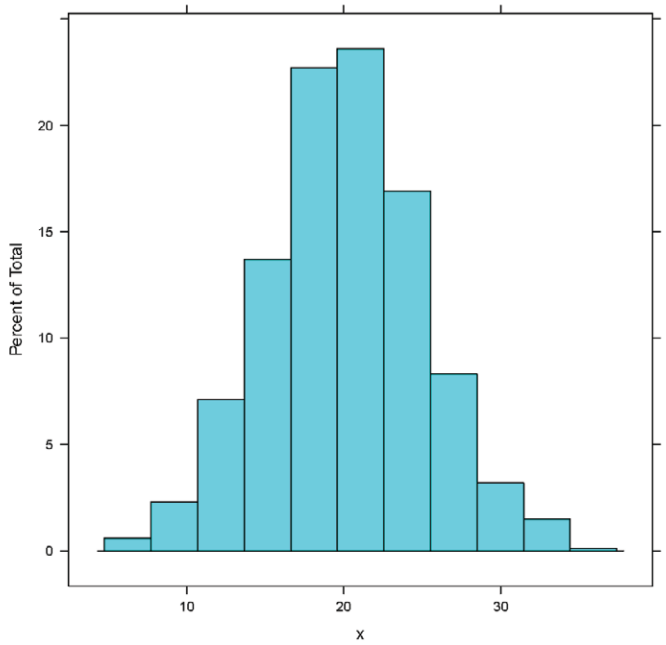


Figure 1: Histogram

Figure 1.18.2: Output in the file *Example.1.18.pdf*

```
par(mar = c(0.5,4,1,4))
boxplot(x)
```

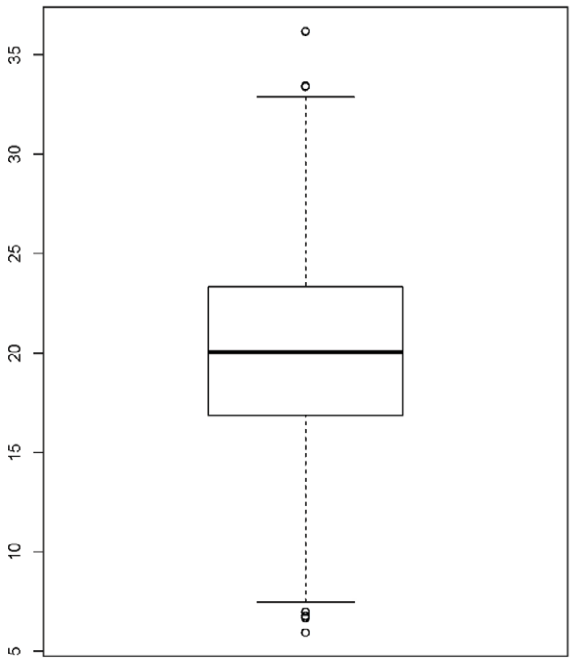


Figure 2: Boxplot

Figure 1.18.3: Output in the file *Example.1.18.pdf*

```
## two plots side by side (option fig.show='hold')
par(mar=c(4,4,.1,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,.7,0),tcl=-.3,las=1)
boxplot(x)
hist(x,main='')
```

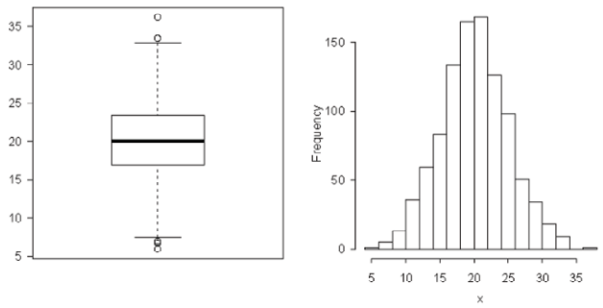


Figure 3: Boxplot and Histogram side-by-side

```
n <- 100
x <- rnorm(n)
y <- 2*x + rnorm(n)
out <- lm(y ~ x)
summary(out)$coef
```

Estimate Std. Error t value Pr(>|t|) (Intercept) -0.02144851 0.10092256
-0.2125244 8.321393e-01 x 2.01895065 0.09660636 20.8987346 6.896270e-38

```
kable(summary(out)$coef, digits=2, format = 'latex')
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.02	0.1	-0.21	0.83
x	2.02	0.1	20.90	0.00

Table 1: Estimated regression coefficients

Figure 1.18.4: Output in the file *Example.1.18.pdf*

1.19 Command line editing

Commands given in an R session are stored together with commands given in previous sessions in a file *.History* in the same folder as the *.RData* file. In an R session previous commands can be retrieved at the R prompt by pressing the up and down arrow keys. A previous command can then be edited using the *backspace*, *delete*, *home*, *end* keys as well as the shortcuts for *copy* and *paste*.

2 Managing objects

After completing the introductory chapter you now know how to

- initialize an R session;
- save your workspace (working directory);
- open an existing workspace;
- executing simple tasks in R to obtain numerical, text or graphical results;
- obtain help.

You know also that everything in R can be considered as some kind of an object. In this chapter the focus is on what properties the different objects have and how to manage objects in the workspace.
However, before we turn our attention to objects we first take a look at managing workspaces. In general it is ***good practice to have each project in its own workspace***. In addition to the methods to create more than one workspace introduced in Chapter 1 we will start this chapter by learning how to use the R functions `getwd()`, `setwd()` and `save.image()` for managing our various workspaces.

2.1 Creating separate workspaces for each project

a) Resume your R session of Chapter 1 by finding out where the .RData folder of your current workspace exists on your computer using the instruction

```
> getwd()↲
```

b) Your next task is to save this workspace in the subfolder *Chapter_1* of the folder *R.Module* on a memory stick. Follow these steps:

```
> setwd("F:\\R.Module\\Chapter_1" ) ↲ # Use appropriate
                                      #drive letter
```

Note: for the above command to work correctly the folder *R.Module* with the subfolder *Chapter_1* MUST EXIST PRIOR to the `setwd()` command. Check if anything has been created in *F:\R.Module\Chapter_1*. Now enter

```
> save.image() ↲
```

and check that the file *.RData* appears in *F:\R.Module\Chapter_1*.

- If the `save.image()` command is not given the *.RData* exists only temporarily and is not visible.
- The `save.image()` command is shorthand for the more general purpose `save()` command to be discussed in a later chapter.

c) Create an empty workspace for Chapter 2 located in subfolder *Chapter_2* of the folder *R.Module* on your memory stick form the console with the instructions:

```
> setwd("F:\\R.Module\\Chapter_2" ) ↲
> getwd()↲ # To check that you are in the correct folder.
> rm(list = objects())↲ # To remove all objects
> save.image() ↲ # To create (save) the empty .RData
```

2.2 Instructions and objects in R

2.2.1 General

Recall that

- instructions are separated by a semi-colon or start on new lines;
- the # symbol marks the rest of the line as comments. See above;
- the default R (primary) prompt is > ; the secondary default prompt is + ;
- use of <- to create objects. (In R version 3.1.x the equality sign (=) will also be accepted. However, avoid this practice and use
  - = only for function arguments;
  - <- for assignment and
  - == for comparison / control structures);
- the use of -> for assigning left-hand side to the name on right-hand side.
- the use of function `assign`() for assigning names to objects. (to be discussed in detail in Chapter 3)

Examples

```
> aa <- 1:10
```

Assigning numeric vector to name "aa". Assignment takes place in global environment.

```
> Aa <- seq(from = 1,to = 10,by = 0.01); yy <- c("a","b","c")
```

```
> c("a","b","c") -> bb
```

Assigning character vector to name "bb".

```
>assign("aa", rnorm(10), pos = 1)
```

Note the use of the argument pos. " " or ' ' are used for characters. Be careful when mixing single quotes and double quotes. See below.

```
> c("u",'v',"'w'",""x"",'"y"',''z'') -> cc
Error: unexpected symbol in "c("u",'v',"'w'",""x"
```

```
> c("u",'v',"'w'",'"x"','"y"',''z'') -> cc
Error: unexpected symbol in "c("u",'v',"'w'",'"x"','"y"',''z"

> c("u",'v',"'w'",'"x"','"y"','z') -> cc
> cc
[1] "u" "v" "'w'" "\"x\"" «\»y\"" "z"
```

- Explain error message above.
- Explain backslash above.

```
> objects()
[1] "aa" "Aa" "bb" "cc"
[5] "yy"
> aa
[1] 1.1849662 -0.2486378 0.9364591 -0.7281662 0.2323667 [6] 0.6910525
-1.2229246 -0.6628765 -1.0274450 -1.1747854

> bb
[1] "a" "b" "c"
```

```
> objects()[3]
[1] "bb"

> parse(text=objects()[3])
 expression(bb)

> eval(parse(text=objects()[3]))
[1] "a" "b" "c"

> rm(a,b)
Warning messages:
1: In rm(a, b) : object 'a' not found
2: In rm(a, b) : object 'b' not found

> rm(aa,bb)

> objects()
[1] "Aa" "cc" "yy"

> rm("cc")

> objects()
[1] "Aa" "yy"
```

2.2.2 Objects in R (or S-PLUS)

a) Everything is an object but there are many different types of objects.

b) Study and also take note of the following ***naming conventions***:

- Allowed are upper or lower case letters, numbers 0–9, full stop(s) and underscore(s).
- Must not begin with a number.
- R is case sensitive i.e. `John` and `john` refer to different objects.
- Use full stops (periods) or underscores to break up a name into meaningful words.
- Avoid `c, s, t, C, F, T, diff` as well as other reserved words for naming an object.

c) The use of the functions `conflicts()` and `find()` when naming objects. The instruction `conflicts(detail=TRUE)` outputs details on whether and where objects with identical names exist on the search path e.g.

```
> conflicts(detail=TRUE)
```

```
$`package:methods`
[1] "body<-" "kronecker"
$`package:base`
[1] "body<-" "kronecker"
```

The instruction `find("object")` outputs details on whether and where objects with the name `object` exist on the search path e.g.

```
> find("kronecker")
[1] "package:methods" "package:base"
```

d) Objects can possess several ***attributes*** e.g.

- mode (The way an object is internally stored)
- length
- names
- dim
- class

Examples

```
> a <- 1:10

> class(a)
[1] "integer"

> b <- factor(c("a","b","c"))

> class(b)
[1] "factor"

> b
[1] a b c
Levels: a b c
```

Levels show that it is a categorical variable (object)

```
> mode(a)
[1] "numeric"

> mode(b)
[1] "numeric"
```

This tells us that the categorical variable (object) `b` is internally stored as a set of numeric codes.

```
> length(a)
[1] 10

> length(b)
[1] 3

> dim(a)
NULL

> mat <- matrix(1:12,nrow=4)

> mat
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
> dim(mat)
[1] 4 3

> mode(mat)
[1] "numeric"

> logic <- c(TRUE,TRUE,FALSE,TRUE)

> mode(logic)
[1] "logical"

> class(logic)
[1] "logical"
```

e) Objects in R are ***vectors***, ***functions*** or ***lists***. There are no scalars – instead vectors of length one are used. In addition to the above three types, there are several other types of objects.

f) Objects that are created during a session are permanently stored in the *.RData*

g) file in the folder containing the workspace (unless not saved at termination).

h) Objects that are created within a function exist only for as long as the function is being executed.

i) Use of `rm()` and `rm(list = ListOfNames)` to remove objects from the workspace.

j) Use of `objects()` or equivalently `ls()` to obtain a list of object names in a data base (by default the workspace). Note the optional arguments `pos, all.names` and `pattern` to specify which database to be considered and what object names to include.

k) How can an object be printed to the screen?

l) ***Warning***: If a new object is assigned to a name that already exists in the working directory the old object is over written without warning and it cannot be retrieved again.

2.2.3 Importing data

a) R has several built-in data sets. Use `>?datasets` and/or `>library(help="datasets")` for details. Note that the two instructions return different information.

b) Study the help file of `c()`.

c) Study the help file of `scan()`.

d) Study the help files of `read.table()` and `read.csv()`. Care must be taken with data containing characters (text) and categorical variables.

e) Using Excel to read in data:
- Prepare data in Excel.
- Make sure there are no empty cells.
- Make sure text values do not contain spaces or special characters.
- Copy to clipboard.
- In R:

```
> objectname <- read.table(file = "clipboard") ↲
```

- Use `h=TRUE` if the first row in the Excel file contains column headings.

```
> objectname <- read.table(file = "clipboard", h=TRUE) ↲
```

2.2.4 Generation of data

Study the operators and functions `:`, `seq()`, `rep()`, `rev()`, `rnorm()`, `runif()` with the following instructions:

```
> 1:10 ↲
> 8:3 ↲
> seq(from=1, to=10, length=10) ↲
> seq(from=2, to=10, length=5) ↲
> rev(10:1) ↲
> rnorm (20, mean=50, sd=5) ↲
> runif (10, min=1, max=3) ↲
```

The function `rmvnorm()` for generating multivariate normal samples is in the `mvtnorm` R package. This package must first be loaded by using the instruction
`>library(mvtnorm)` ↲ or *load* from the main menu.
Alternatively, for generating multivariate normally data there is also a function `mvrnorm()` in R package `MASS`.

2.3 How R finds data

In order to understand how objects are found by R it is necessary to have some understanding of the concepts

- Environment
- Frame
- Search path
- Parent environment
- Inheritance.

The mechanism that R uses to organize objects is based on frames and environments. A *frame* is a collection of named objects and an *environment* consists of a frame together with a pointer or reference to another environment called the *parent environment*. Environments are nested so that the parent environment is the environment that directly contains the current environment. At the start of an R session a workspace is created which always has an associate environment, the *global environment*. The global environment occupies the first position on the *search path* and is accessed by a call to `globalenv()`. Packages and databases can be added to the search path by a call to `attach()` and removed from the search path by a call to `detach()`.

- What is an R *package*? What is the difference between *installing* and *loading* a package?
- Work through the following example:

```
> search()
```

to obtain the search path

```
[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:utils" "package:datasets"
[7] "package:methods" "Autoloads" "package:base"
```

```
> library(MASS)
```
to attach the search path (`attach()` used for dataframes)

```
> search()
```
`MASS` attached by default in second position
```
 [1] ".GlobalEnv" "package:MASS" "package:stats"
 [4] "package:graphics" "package:grDevices" "package:utils"
 [7] "package:datasets" "package:methods" "Autoloads"
 [10] "package:base"
```

```
> detach("package:MASS")
```
remove `MASS` from the search path
```
> search()
[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:utils" "package:datasets"
[7] "package:methods" "Autoloads" "package:base"
> parent.env(.GlobalEnv)
```
to obtain the parent of the global environment
```
<environment: package:stats>
attr(,"name")
[1] "package:stats"
attr(,"path")
[1] "C:/Program Files/R/R-3.1.0/library/stats"

> environmentName(parent.env(.GlobalEnv))
[1] "package:stats"

> parent.env(parent.env(.GlobalEnv))
<environment: package:graphics>
attr(,"name")
[1] "package:graphics"
attr(,"path")
[1] "C:/Program Files/R/R-3.1.0/library/graphics"
environmentName(parent.env(parent.env(parent.env(.GlobalEnv))))
[1] "package:grDevices"
```

When the R evaluator looks for an object and it cannot find the name in the global environment it will search the parent of the global environment. It will carry on the search along the search path until the first occurrence of the name. If the name is not found it will return the message

```
Error: object 'xx' not found
```

The usage of the double colon `::` and the triple colon `:::` is to access the intended object when more than one object with the same name exist on the search path. These two operators use the namespace facility of R packages. The namespace of a package allow the creator of a package to hide functions and data that are meant only for internal use; it provides a way through the operators `::` and `:::` to an object within a particular package. Thus a namespace prevent functions from breaking down when a user selects a name that clashes with one in the package. The double-colon operator `::` selects objects from a particular namespace. Only functions that are exported from the package can be retrieved in this way. The triple-colon operator `:::` acts like the double-colon operator but also allows access to hidden objects. Packages are often inter-dependent, and loading one may cause others to be automatically loaded. Such automatically loaded packages are not added to the search list.

We note that the function call `getAnywhere()`, which searches multiple packages can be used for finding hidden objects.

When a ***function*** is called, R creates a new (temporary) environment which is enclosed in the current (calling) environment. Objects created in the new environment are not available in the parent environment and dies with it when the function terminates. Objects in the calling environment are available for use in the new environment created when a function is called.

Similarly, when an ***expression*** is evaluated a hierarchy of environments is created. Search for objects continue up this hierarchy and if necessary to the global environment and from there up onto the search path.

- Study the use of the arguments `pos`, `all.names` and `pattern` of the function `objects()`.
- Study the behaviour of the functions `conflicts()` and `exists()` in the examples below:

```
> conflicts()
[1] "body<-" "kronecker"

> conflicts(detail=TRUE)
$'package:methods'
[1] "body<-" "kronecker"
$`package:base`
[1] "body<-" "kronecker"

> exists("kronecker")
[1] TRUE
```

```
> exists("kronecker", where=1)
[1] TRUE

> exists("kronecker", where=1,inherits=FALSE)
[1] FALSE

> exists("kronecker", where=2)
[1] TRUE

> exists("kronecker", where=2,inherits=FALSE)
[1] FALSE

> exists("kronecker", where=7,inherits=FALSE)
[1] TRUE

> exists("kronecker", where=8,inherits=FALSE)
[1] FALSE

> exists("kronecker", where=9,inherits=FALSE)
[1] TRUE
```

- Study the above code carefully and then explain what inheritance does.
- The example below leads to the same conclusion as above but is more complicated at this stage. Its behaviour will become clear as we work through the coming chapters.

```
> sapply(search(),function(x)exists("kronecker",
  where=x,inherits=FALSE))
            .GlobalEnv     package:stats   package:graphics
                 FALSE             FALSE             FALSE
      package:grDevices     package:utils   package:datasets
                 FALSE             FALSE             FALSE
        package:methods         Autoloads       package:base
                  TRUE             FALSE              TRUE
```

- Direct access to objects down the search path can be achieved with the function `get()`. The function `get()` takes as its first argument the name of an object as a character string. The optional argument `pos` can be used to specify where on the search list to look for the object. As an illustration explain the outcomes of the following function calls:

```
> get("%o%")
> mean <- mean(rnorm(1000))
> get(mean)
> get("mean")
> get("mean", pos = 1)
> get("mean", pos = 2)
> get("mean", pos = 2, inherits = FALSE)
```

- Instead of attaching databases the function `with()` is often to be preferred. Discuss the usage of `with()` by referring to the instructions
  1. `with(beaver1,mean(time))` and
  2. `with(beaver2,mean(time))`, respectively.

2.4 The organization of data (data structures)

Study carefully the help files of `list()`, `matrix()`, `data.frame()` and `c()`.
A ***list*** is created with the function `list()`. A list is the basic means of storing a collection of data objects in R when the modes and/or lengths of the objects are different. List elements are accessed using [[ ]] or $ when the objects are named. List objects are named using the construction

```
> my.list <- list(name1=object1, name2=object2)
```

and elements are retrieved using the instruction

```
> my.list[[2]] or
> my.list$name2.
```

A ***matrix*** in R is a rectangular collection of data, all of the same mode (e.g. numeric, character/text or logical). It is formed with the construction

```
> my.matrix <- matrix(x, ncol=n, nrow=n, byrow=FALSE)
```

Matrix elements are accessed using `my.matrix[i,j]`. The functions `nrow()`, `ncol()`, `dim()`, `dimnames()`, `colnames()` and `rownames()` are useful when working with matrices.

A ***dataframe*** is also a rectangular collection of data but the columns can be of different modes. It can be regarded as a cross between a list and a matrix. Dataframes are constructed with the function `data.frame()`.

Study the help files of the above functions. The function `read.table(file="c:\\full path", h=TRUE)` reads the contents of an external file into a dataframe. Study the help file for a full understanding of its usage.

2.5 Time series

Study the usage of the function `ts()`.

2.6 The functions `as.xxx()` and `is.xxx()`

The function `as.xxx()` transforms an object as best as possible to a specified type e.g.
`as.matrix(mydata)` transforms the numerical dataframe to a numerical matrix.
`is.xxx()` tests if the argument is of a certain type e.g. `is.matrix(mydata)`
evaluates to `FALSE` if `mydata` does not satisfy all the conditions of a matrix.

2.7 Simple manipulations; numbers and vectors

- Explain vector calculations and the recycling principle by referring to the example below.

```
> c(1,3,5,9) + c(1,2,3)
[1] 2 5 8 10
Warning message:
In c(1, 3, 5, 9) + c(1, 2, 3) :
longer object length is not a multiple of shorter object length
```

- Logical vectors. Explain the behaviour of the instruction below

```
> sum(c(TRUE,FALSE,TRUE,TRUE,FALSE))
[1] 3
```

- Missing values: NA (indicate a missing value in the data), NaN (not a number)

```
[1] 3
> 10/0
[1] Inf
> 0/0
[1] NaN
```

- Character vectors: see section 3.6
- Subscripting vectors: see section 5.1

2.8 Objects, their modes and attributes

- Vectors elements must be of same mode: logical, numeric, complex, character, raw
- Empty object; once created (e.g. `xx <- numeric()`) components may be added (e.g. `xx[5] <- 22`)
- Getting and setting attributes: The functions `attr()` and `attributes()`
- Class of an object and the function `unclass()` for removing class.

2.9 Representation of objects

We have seen already that a representation of an object can be obtained by calling (entering) its name:

```
> cars
    speed dist
1     4      2
2     4     10
3     7      4
. . . . . . .
49    24    120
50    25     85
```

It is often not convenient to have a full representation returned of an object as above. The functions `head()`, `str()` and `summary()` are available for extracting a partial representation of an object:

```
> head(cars)
   speed dist
1     4      2
2     4     10
3     7      4
4     7     22
5     8     16
6     9     10

> summary(cars)
        speed                  dist
 Min.      : 4.0      Min.       : 2.00
 1st Qu.   :12.0      1st Qu.    : 26.00
 Median    :15.0      Median     : 36.00
 Mean      :15.4      Mean       : 42.98
 3rd Qu.   :19.0      3rd Qu.    : 56.00
 Max.      :25.0      Max.       :120.00

> str(cars)
'data.frame': 50 obs. of 2 variables:
 $ speed: num 4 4 7 7 8 9 10 10 10 11 …
 $ dist : num 2 10 4 22 16 10 18 26 34 17 …
```

There are many more R functions provided for getting information of what an R object represents. Some of these functions like `mode()`, `class()`, `length()`, `levels()`, `is.xxx()` and `as.xx()` have been already encountered and others will be given in the chapters to come.

```
> length(cars)
[1] 2
```

Could you explain this?

```
> length(as.matrix(cars))
[1] 100

> dim(cars)
[1] 50 2

> is.matrix(cars)
[1] FALSE

> is.data.frame(cars)
[1] TRUE

> is.list(cars)
[1] TRUE

> mode(cars)
[1] "list"

> class(cars)
[1] "data.frame"
> levels(cars)
NULL
```

Explain.

2.10 Exercise

1. According to the central limit theorem (CLT) the distribution of the sum (or mean) of independently, identically distributed stochastic variables converges to a normal distribution with an increase in the number variables. The binomial distribution can be expressed as the sum of independently, identically distributed Bernoulli stochastic variables and therefore converges in distribution to the normal distribution. The lognormal distribution in contrast cannot be expressed as a sum.

   Make use of the function `rbinom()` to generate a sample of size 10 from a binomial distribution modelling 20 coin flips with a probability of 0.4 for returning "heads". Use the function `hist()` to graph the results. Repeat with sample sizes 50, 100, 1000, 10000 and 100000. Repeat the whole study with a success probability of 0.5, 0.3, 0.1 and 0.05.

   Discuss your findings.
   Now repeat the same exercise using (a) the lognormal distribution with the function `rlnorm()` and (b) the uniform distribution over the interval [10; 25] with the function `runif(min=10, max=25)`. Comment on your findings.

2. Assume that a random sample of size *n* is available from a certain distribution. A *bootstrap sample* is obtained by sampling *with replacement* a sample of size *n* from the given sample. One of the uses of the bootstrap is to obtain an estimate of the standard error of a statistic. For example, a bootstrap estimate of the standard error of $\bar{X}$ can be obtained as follows:
   a) Generate independently of each other *B* bootstrap samples.
   b) Calculate the mean of the *B* bootstrap samples, i.e. calculate $\bar{x}_1^*, \bar{x}_2^*, \ldots, \bar{x}_B^* \bar{x}_1^*, \bar{x}_2^*, \ldots, \bar{x}_B^*$.
   c) Calculate $\bar{\bar{x}} = \frac{1}{B}\sum_{i=1}^{B} \bar{x}_i^*$.
   d) Calculate $\widehat{se}_B = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}(\bar{x}_i^* - \bar{\bar{x}})^2}$.
      i. Generate a random sample of size 25 from a normal (100; 255) distribution.
      ii. Use R to obtain graphical representations and statistics of the characteristics of the sample.
      iii. Program the necessary instructions in R to obtain bootstrap estimates of the standard error of the *sample mean* as well as the *sample median*. Use 50, 100, 500 and 1000 for B (the number of bootstrap repetitions). How do your answers compare with what is *theoretically* expected?
      iv. Program the necessary R instructions to obtain graphical representations of the bootstrap distribution in (iii).

3. Generate a random sample of size 50 from a multivariate normal distribution with mean vector (118, 396, 118, 400) and a covariance matrix so that the variances of the variables are given by 778, 1810, 580 and 2535 respectively. Variables 1 and 2 have a covariance of -642.5 and variables 3 and 4 have a covariance of -670. The other variables are uncorrelated. Store the sample as a matrix object and then program the necessary R instructions to calculate the sample covariance matrix and sample mean vector.

4. Execute the instruction
   `> set.seed(101023)` ↵
   Next, obtain 400 random normal (0; 1) values and arrange them in a matrix with 20 rows and 20 columns. Finally, write and R function to calculate and return (i) the sum of all the elements in the matrix, (ii) the eigenvalues of the matrix, (iii) the inverse of the matrix as well as (iv) the rank of the matrix. Hint: Read the help of the functions `eigen()` and `solve()`.)

3 R operators and functions

After completing Chapters 1 and 2 it is assumed that the following are now familiar:

- How to communicate with R;
- How to manage workspaces;
- How to perform simple tasks using R.

In this chapter we take a closer look at the behaviour of some of the most common

- R operators
- R functions.

3.1 Arithmetic operators

a) Study the use of the operators in Table 3.1.1.

Operator	Function	Operator	Function
+	Addition	^	Exponentiation
−	Subtraction	%/%	Integer divide
*	Multiplication	%%	Modulus
/	Division	:	Sequence
%*%	Matrix multiplication	−	Unary minus

Table 3.1.1: Arithmetic operators.

Note that the arithmetic operators are also functions. That this is so follows by studying the following examples:

```
> 3+7
[1] 10
> "+"(3,7)
[1] 10
> 17 %% 3
[1] 2
> "%%"(17,3)
[1] 2
```

b) Rules for operator expressions with vector arguments.
Study the results of the following R instructions

1. `cars [,2] * 12 * 25.4 / 1000`

Interpret the result

2. `7%/%3`

3. `7%%3`
4. `matrix(1,nrow=4,ncol=4) * matrix(3,nrow=4,ncol=4)`
5. `matrix(1,nrow=4,ncol=4) %*% matrix(3,nrow=4,ncol=4)`

Explain the following instructions and output from R:

```
> 1:12 + 1:3
 [1]  2  4  6  5  7  9  8 10 12 11 13 15
> 1:10 + 1:2
 [1]  2  4  4  6  6  8  8 10 10 12
> 1:10 + 1:3
 [1]  2  4  6  5  7  9  8 10 12 11
Warning message:
In 1:10 + 1:3 :
longer object length is not a multiple of shorter object length
```

In the above examples it is illustrated that R uses ***vectorized arithmetic*** i.e. it operates on vectors as wholes. Sometimes the ***recycling principle*** is applied with or without a warning. It is a good R programming habit to make use of vectorizing calculations where possible. The effect of the recycling principle must be kept in mind since it might lead to unwanted results.

c) Missing values, infinity and "not a number".

A missing value in R is denoted by NA. ***The result of a computation involving NAs is always NA*** *e.g.*

```
> mean(c(1,3,NA,12,5))
[1] NA
```

The result of a computation that cannot be represented as a number e.g. 0/0 is denoted by `NaN`:

```
> 0/0
[1] NaN
```

Note: some computational results are differently reported by R as the corresponding algebraic equivalents e.g.

```
> 5/0
[1] Inf
```

Undefined in algebra

```
> -5/0
[1] -Inf
> 5/(-0)
[1] -Inf
```

5/0 in R is given by `Inf` while algebraically it is undefined.

d) Scientific notation

R uses decimal notation as well as scientific notation for arithmetic calculations:

```
> 60000000
[1] 6e+07
```

6×10^7

```
> 1/6000000
[1] 1.666667e-07
```

1.666667×10^{-7}

Scientific notation is not to be confused with e^x:

```
> exp(15)
[1] 3269017
> exp(-15)
[1] 3.059023e-07
```

e) How are numbers represented in a computer's memory? What are the implications of this?
Computers use ON/OFF (or 1/0) switches for encoding information. A single switch is called a ***bit*** and a group of eight bits is called a ***byte***. A single integer is represented exactly in a computer by a fixed number of bytes i.e. 32 or 64 bits. There are several schemes according to which integers are represented by bits in a computer. This representation in a computer takes place at a level where R has no control over it but R stores information about the computing environment in an object `.Machine`. The element `.Machine$integer.max` returns the largest integer that can be represented in the computer on which R is running e.g.

```
> .Machine$integer.max
[1] 2147483647
```

Although the above method of representing integers by strings of bits provides a very efficient way of storing integers in a computer R usually treats integers similar to real numbers by using ***floating point representation***. In binary floating point notation a number x is written as a sequence of zeros and ones (the ***mantissa***) times two with an exponent say m: $x = b_0 b_1 b_2 \ldots \times 2^m$ where $b_0 = 1$ except when $x = 0$.
In practice there is only a limited number of b's available and the exponent is also limited therefore, in general, not all real numbers can be represented exactly in a computer – they can at most be approximated. The smallest number x such that $1 + x$ can be distinguished from 1 in a computer is called ***machine epsilon***. In R this can be obtained from `.Machine$double.eps` e.g.

```
> .Machine$double.eps
[1] 2.220446e-16
```

Although floating point representation allows computation with very small (in magnitude) and very large numbers the above limitations can lead to ***underflow*** (a result of 0) or ***overflow*** (a result of $\pm\infty$) which can have disastrous consequences in practice. Writing good code in R must take the above seriously into account.

3.2 Logical operators

Logical operators result in `TRUE`, `FALSE` or `NA`. Study the use of the logical operators in Table 3.2.1.
Warning: While it is perfectly legitimate to write

```
> x[x == -1] <- 0 or > x[x == 1] <- 0
```

it is incorrect to specify

```
> x[x == NA] <- 0 or > x[x == NaN] <- 0
```

The correct code in the latter case is

```
 > x[is.na(x)] <- 0 or > x[is.nan(x)] <- 0
```

What are the consequences of the above code? Also take note of the functions `any()` and `all()`. These two functions are useful when combining logical objects. Give the necessary instructions to carry out the following tasks:

a) Check if any of the states in the `state.x77` data set have populations with an illiteracy rate that is not larger than 1.6 and a Murder rate of more than 10.0.

b) Check if there is at least one state with income greater than $5000 and life expectancy less than 70.0 years.

c) Check if all states with an income of more than $5000 has an illiteracy of below 2.0.

What is meant by a control logical operator?

Operator	Function	Operator	Function
>	Greater than	&	Elementwise and
<	Less than	|	Elementwise or
<=	Less than or equal to	&&	Control and
>=	Larger than or equal to	||	Control or
==	Equality	!	Unary not
!=	Non-equality		
+	Addition	^	Exponentiation

Table 3.2.1: Logical operators.

d) Carry out the instructions:

```
mata <- matrix(1:4,ncol=2)
matb <- matrix(c(10, 20,30,40), ncol=2)
mata; matb; mata>1 & matb>1; mata>1 | matb>1;
mata>1 && matb>1; mata>1 || matb>1
```

Comment on the above.

e) What is the result of

```
> sum(c(TRUE, !FALSE, FALSE, TRUE, TRUE)) ?
```

f) What is the result of

```
> sum(c(TRUE, !FALSE, FALSE, NA, TRUE)) ?
```

Explain.

3.3 The operators <-, <<- and ~

Before considering the use of these operators answer the following:

a) What will happen to an object `aa` in the working directory if within a function the following assignment is made

```
aa <- 20?
```

b) Now, study the help file of <<− and then answer (a) if the operator <− has been replaced with the operator <<−.
Warning: use <<− very carefully.

c) The tilde operator is used in modelling functions, e.g.

```
lm(length ~ age).
```

3.4 Operator precedence

Study the precedence rules as summarized in Table 3.4.1. The rules followed are shown in Table 3.4.1 from top to bottom and left to right. Note the use of

- *parentheses* ( ) for function arguments and changing precedence,
- *braces*{ } for demarcating blocks of instructions
- and *brackets* [ ] for subscripting.

The correct way of extracting the fifth element of a sequence like `1:20` is

```
> (1:20)[5]
```

Operator	What it does
`$`	List and dataframe subscripting
`[], [[]]`	Vector and matrix subscripting; list subscripting
`^`	Exponentiation
`%*%, %/%, %%`	Matrix multiplication; integer divide; modulus
`*, /`	Multiplication and division
`+, -`	Addition and subtraction
`<, >, <=, >=, + =, !=`	Logical comparisons
`!`	Unary not
`&, |, &&, ||`	Logical and; logical or; control and, control or
`<-, <<-`	Assignment

Table 3.4.1: Precedence rules.

Explain the results of the following R instructions:

```
> 20 / 4 * 12 ^2 - 6 + 14
> (20 / 4) * (12 ^2) + (-6 + 14)
> 20 / 4 * 12 ^(2 - 6 + 14)
> 20 / 4 * (12 ^2 - 6 + 14)
```

3.5 Introduction to functions in R

A function in R consists of

i. the keyword ***function***, followed by

ii. parentheses () enclosing the ***arguments*** of the function with

iii. the ***body*** of the function (instructions or lines of code) appearing in braces {}.

The arguments of a function can be inspected by using the command

```
> args(name of function)
```

The function `str(x)` provides information on the object *x*. If *x* is a function its output is similar to that of `args()`. ***Default values*** are given to function arguments using the construction (argument `name = value`). It is good programming practice to make extensively use of comments to describe arguments and / or what a particular chunk of code does.

What is the usage of the following function:

```
> cube <- function(a) a^3
```

In the above function the argument `a` is called a ***dummy argument***. What will happen to an object `a` in the working directory?

Functions are called by replacing the ***formal arguments*** by the ***actual arguments***. This can be done ***by position*** or ***by name***. *Hint*: It is less error prone to call functions using named arguments. Create the following function

```
> Demofunc <- function(vec = 1:10, m,k)
+ { # Function to subtract a specified constant from
+   # each element of a given vector and after subtraction
+   # divide each element by a second specified constant.
+   # The result of the above transformation is returned.
+ (vec - m)/ k
+ }
```

Execute the following function calls and explain the output

```
> Demofunc(3, 2, 5)
> Demofunc(2,5)
> Demofunc(m = 2, k = 5)
> Demofunc(m = 2, k = 5, vec = 1:100)
```

Note the use of `prompt()` and `package.skeleton()` to provide a new function with a helpfile.

3.6 Some mathematical functions

3.6.1 General mathematical functions

```
abs(),exp(),log(x, base = exp(1)),log10(),gamma(),sign(), sqrt()
```

3.6.2 Trigonometric functions

See Table 3.6.1.

Operator	Function	Operator	Function
`cos()`	Cosine	`acos()`	arc cosine
`sin()`	Sine	`asin()`	arc sine
`tan()`	Tangent	`atan()`	arc tangent
`cosh()`	hyperbolic cosine	`acosh()`	arc hyperbolic cosine
`sinh()`	hyperbolic sine	`asinh()`	arc hyperbolic sine
`tanh()`	hyperbolic tangent	`atanh()`	arc hyperbolic tangent

Table 3.6.1: Trigonometric functions.

3.6.3 Complex numbers

```
Arg(), Conj(), Mod(), Re(), Im()
```

3.6.4 Functions for rounding and truncating

`round(), ceiling(), floor(), trunc()`

Study the help files of the above functions. Check all arguments.

3.6.5 Functions for matrices

Study Table 3.6.2 in detail.

Two other functions that play an important role in matrix calculations are the functions `rbind()` and `cbind()` for concatenating matrices row-wise or column-wise.

Revise also the functions `matrix()`, `dim()`, `dimnames()`, `colnames()`, `rownames()` as well as `scan()` and `read.table()`.

Function	What it does
`chol()`	Cholesky decomposition
`crossprod()`	Matrix crossproduct
`diag()`	Create identity matrix, diagonal matrix or extract diagonal elements depending on its argument
`eigen()`	Finding eigenvectors and eigenvalues
`kronecker()`	Computing the kronecker product of two matrices
`outer()`	Outer product of two vectors
`scale()`	Centring and scaling a data matrix
`solve()`	Finding the inverse of a nonsingular matrix
`svd()`	Singular value decomposition of a rectangular matrix
`qr()`	QR orthogonalization
`t()`	Transpose of a matrix

Table 3.6.2: Functions for matrices.

a) The function `chol()` performs a Cholesky decomposition of the square, symmetric, positive definite matrix $\mathbf{A} = \mathbf{U}'\mathbf{U}$ where $\mathbf{U}$ is an upper triangular matrix.

b) The function `crossprod (A, B)` returns the matrix $\mathbf{A}'\mathbf{B}$.

c) The function `diag(arg)` does various things depending on its argument: if `arg` is a positive integer `diag(age)` returns an identity matrix of the given size; if `arg` is a vector `diag(arg)` returns a diagonal matrix with diagonal elements the respective elements of the given vector; if `arg` is a matrix then `diag(arg)` returns a vector containing the diagonal elements of the given matrix.

d) What is the difference between `diag(A)` and `diag(diag(A))`?

e) The function `eigen()` operates on a square matrix and returns a list with named elements values and vectors containing respectively, the eigenvalues and eigenvectors. Study the help file of `eigen()` carefully.

f) The function `kronecker()` returns the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ of matrices **A** and **B**.

g) The function `outer (x, y, f)` operates on two vectors ***x***: $n \times 1$ and ***y***: $p \times 1$ to return a matrix of size $n \times p$ with *ij* th element the result of applying the function `f` on `x[i]` and `y[j]`. The default for `f` is "`*`".

h) The function `scale()` has three arguments: a matrix as first argument; a second argument `center` and a third argument `scale`. If `center = FALSE`, no centring of the columns of the matrix argument is performed, if set to `TRUE`, the mean value of each column is subtracted from the respective columns, if given a vector of values these values are subtracted from the respective columns. If `scale = FALSE`, no scaling of the columns of the matrix argument is performed, if set to `TRUE` each column is divided by its standard deviation, if given a vector of values then each column is divided by the corresponding value.

i) The function `solve (A, b)` is used for solving the equation A$\boldsymbol{x}$ = $\boldsymbol{b}$ for $\boldsymbol{x}$, where `b` can be either a vector or a matrix with `A` being a square matrix. If argument `b` is missing it is taken to be the identity matrix so that the inverse of argument `A` is returned.

j) The function `svd()` returns the singular value decomposition of its matrix argument **A** = **UDV′**. It returns a list with three components: u the orthogonal or orthonormal matrix **U**; d the vector containing the ordered singular values of the rectangular matrix ; v the orthogonal or orthonormal matrix **V**.

k) The function `qr()` performs a QR decomposition of any arbitrary matrix $\mathbf{M = QR}$ with **Q** and orthogonal matrix and **R** an upper triangular matrix. Study the help file of `qr()` for full details and usages of `qr()`. Note that the matrices **Q** and **R** can be obtained directly by calling `qr.Q(qr())` and `qr.R(qr())`, respectively.

l) What is the meaning of each of the following instructions?

```
rbind(a,b); rbind(1,x); rbind(a = 1:5,b = 10:14,c=20:24);
cbind( a= 1:5, b=10:14, c=20:24)
```

m) Write a function to calculate the determinant of a square matrix. Name this function `det.own()` in order to distinguish it from the built in R function `det()`.

n) When the user is satisfied with a function, it is often necessary to have it available for all R projects. It is useful to assign all such functions to the same data base or directory. Use the function `assign (x, object, pos = , envir = )` to store the function `det.own()` in your own R functions directory. The argument x in assign() is a ***character string*** for assigning a name to the object. The function `remove (list of objects names, pos = , envir = )` can be used to remove objects from your own or any other database. *Hint*: First create a file and then use attach() to add it to the R search path.

```
> save(file= " C:\\MyFunctions").
```

Study how `save()` works.

```
> attach("C:\\MyFunctions", pos=2).
```

Study how `attach()` works.

```
> assign("det.own", det.own, pos=2).
```

Study how `assign()` works.

```
> save(list=objects(2), file = "C:\\MyFunctions").
```

Explain the use of the argument `list=objects(2)`.

To summarize: The construction

```
> NAME <- object
```

is a simple way to assign an object to a name. This form of assignment always takes place in the global environment (the workspace). Assignment can also be performed using the functions `save()` and `assign()` as illustrated above. The latter form of assignment is more complicated but the assignment is not restricted to the global environment.

o) The result of the function `gamma(x)` is $(x - 1)!$ If x is a non-negative whole number. Now write a function `fact()` to calculate $x!$. This function must make provision for 0! As well as for a negative number or a fraction that is read in by mistake. *Hint*: First study the usage of the *if* statement by requesting help `?Control`, recall Table 1.6.1. Store this function in your directory of R functions. How will you go about to make `fact()` and `det.own()` available for any R project?

p) The function lgamma(x) returns the logarithms of gamma(x). Write a function to calculate the value of $f(n) = \Gamma(\frac{n-1}{2})/\{\Gamma(\frac{1}{2})\Gamma(\frac{n-2}{2})\}$. Calculate the value of $f(n)$ for $n = -10, 10, 100, 500,$ and 1000.

3.6.6 Sorting functions

Note the use of the functions `sort()`, `order()` and `rank()`. First construct `MatX` using the functions `scan()` and `matrix()`. Explain in detail what `order()` does by sorting all the columns of `MatX` according to the values in the first column of the matrix.

$$MatX = \begin{bmatrix} 4 & 80 & 12 \\ 5 & 70 & 70 \\ 6 & 30 & 19 \\ 2 & 40 & 80 \\ 4 & 90 & 40 \\ 1 & 60 & 50 \\ 7 & 10 & 20 \\ 3 & 30 & 200 \end{bmatrix}$$

3.6.7 Some functions for data manipulation

Study the functions in Table 3.6.3.

Function	What it does
`append()`	Combine vectors; more flexibility than c()
`c()`	Create vectors
`duplicated()`	Extract duplicated values
`match()`	Match values in pairs of vectors
`pmatch()`	Partial matching
`replace()`	Replace specified values in vectors
`unique()`	Extract unique values

Table 3.6.3: Functions for data manipulation.

a) Insert the vector (101, 102, 103, 104, 105) into the vector (10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20) after its fifth element by utilising the argument `after` of the function `append()`.

b) The function `replace()` requires three arguments `x`, `list` and `values`. The values in x with indices given in `list` is replaced by the successive values in `values` making use of the recycling principle if needed. Explain this by replacing in the vector (10, 2, 7, 20, 5, 8, 9, 20, 9, 1, 15), the values 10, 20 and 15 with zeros.

c) Find the unique values in the vector (10, 2, 7, 20, 5, 8, 9, 20, 9, 1, 15).

d) Find the duplicated values in the vector (10, 2, 7, 20, 5, 8, 9, 20, 9, 1, 15, 20, 20, 15).

e) Explain the usage of match() by considering the difference between

```
match (c(10,2,7,20,5,8,9,20,9,1,15), c(10,20,15))
match (c(10,20,15), c(10,2,7,20,5,8,9,20,9,1,15))
```

f) Illustrate the difference between `match()` and `pmatch()` by considering the names of the days of the week.

3.6.8 Basic statistical functions

Study in detail the functions in Table 3.6.4.

Function	What it does	Comments
`cor()`	Correlation	One or two arguments
`cumsum()`	Cumulative sum of elements of a vector	
`mean()`	Arithmetic mean	Optional argument trim =
`median()`	Median	Accepts variable number of arguments
`min()`	Min value	Accepts variable number of arguments
`max()`	Max value	Accepts variable number of arguments
`prod()`	Product of elements of a vector	Accepts variable number of arguments
`cumprod()`	Cumulative product of elements of a vector	
`quantile()`	Returns specified quantiles	
`range()`	Min and max of a vector	Accepts variable number of arguments
`sample()`	Random sample	With or without replacement
`sum()`	Arithmetic sum	Also used for counting
`var()`	Variance and covariance; uses $n-1$ as denominator	Accepts vectors or matrices
`sd()`	Standard deviation; uses $n-1$	Accepts a vector as argument

Table 3.6.4: Basic statistical functions.

Note also the functions `pmax()` and `pmin()`.

a) Find the average Life Expectancy of the states in the `state.x77` data set.

b) Find the 5% trimmed mean for Illiteracy of the states in the `state.x77` data set.

c) Find the correlation between the Illiteracy and the Income of the states in the `state.x77` data set.

d) Find the covariance matrix of all the variables in the `state.x77` data set.

e) Find the range for Murder in the `state.x77` data set.

f) Obtain the details of a random sample of 10 states in the `state.x77` data set.

g) Obtain two independent random permutations of the numbers 1, 2, …, 10.

h) Write a function for computing the coefficient of kurtosis for a random sample. Test your function on the Frost variable in the `state.x77` data set.

i) Write a function for computing the coefficient of skewness for a random sample. Test your function on the Murder variable in the `state.x77` data set.

j) Write a function to compute the harmonic mean of a numeric vector. Test your function on the Life Expectancy of the states in the state.x77 data set. Compare your answer to your answer in (a).

3.6.9 Probability distributions in R

First, execute the R-instruction

```
> help.search("distribution")
```

to obtain a list of available statistical distributions in R.

Each distribution has an identifying name preceded by one of the letters *d*, *p*, *q* or *r*. In the case of an F-distribution, for example, the identifier is just the letter `f` and for a normal distribution the identifier is `norm`. Preceding the distribution's identifier by one of the letters `d`, `p`, `q` or `r` returns a density value, a probability, a quantile or a random sample for the specified distribution (probability density function or probability mass function). See Figure 3.6.1 for an explanation.

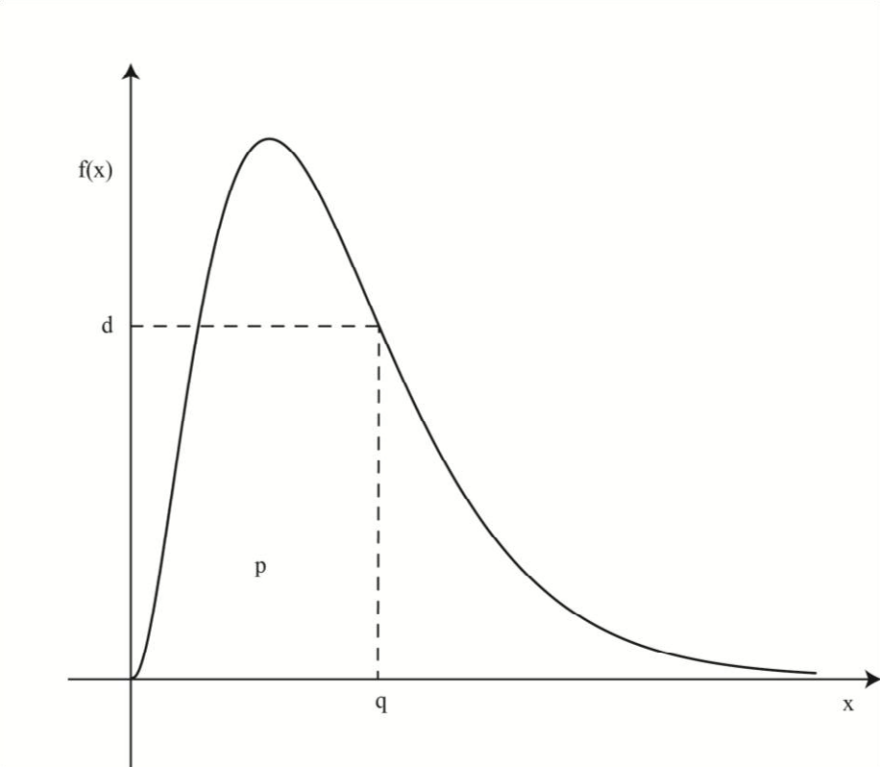


Figure 3.6.1: Meaning of the letters *d*, *p* and *q* when preceding an R distribution identifier.

a) Find a random sample of size 10 from a *normal* (50, $\sigma^2 = 20$) distribution.

b) How can it be arranged that all members in a class obtain the same random sample? Illustrate. *Hint*: Request help on the function `set.seed()`.

3.6.10 Functions for categorical variables

Apart from being *numeric* or *logical*, data in R can also be *categorical* (*factor* in R) or *character strings*. Study in detail the functions operating on factor data in Table 3.6.5.

a) Use cut() to create an object `areagrp` to divide the `state.x77` data set into three groups representing the states with area within the intervals $(0, 10\,000]$, $(10\,000, 100\,000]$ and $(100\,000,$ Inf], respectively. *Hint*: First study the arguments of `cut()`.

b) Repeat (a) with argument `labels = ??` to specify each state as being *Small*, *Medium* or *Large* with respect to its area.

c) Use `unclass()` to obtain the numeric codes associated with each level of `areagrp`.

d) Repeat (a) to obtain `areagrp2` containing five equally spaced categories.

e) Repeat (a) to obtain `areagrp3` containing five groups with each containing 20% of the data.

f) Use `cut()` to create an object `illitgrp` to divide the `state.x77` data set into five groups representing the states with illiteracy within the interval $[0, 0.50)$, $[0.50, 1.00)$, $[1.00, 1.50)$, $[1.50, 2.00)$ and $[2.00, 5.00)$, respectively.

g) Obtain a two-way table of the `state.x77` data set according to `areagrp` and `illitgrp`.

Function	What it does
`cut()`	Creates categories out of a continuous variable
`factor()`	Encodes a vector as a ***nominal*** categorical variable
`factor()`	Encodes a vector as a ***ordinal*** categorical variable when argument ordered is set to TRUE
`levels()`	Displays or sets the levels of a factor variable
`pretty()`	Creates convenient break points for a categorical variable
`split()`	Breaks up an array according to the value of a categorical variable
`table()`	Counts the number of observations cross-classified by categories
`unclass()`	Returns the numeric codes for representing the levels of a factor variable

Table 3.6.5: Basic functions for categorical variables.

3.6.11 Functions for character manipulation

Study the functions in Table 3.6.6 in detail.

Function	What it does
`abbreviate()`	Generates abbreviations of character values
`cat()`	Display,messages and/or values on screen or send to file
`grep()`	Search for patterns in characters
`nchar()`	Number of characters in a string
`paste()`	Combine values into character strings
`strsplit()`	Split the elements of a character vector `x` into substrings
`substring()`	Extracts parts of character strings

Table 3.6.6: Basic functions for character manipulation.

a) What is the returned value of `grep ("ia", state.name)`?

b) Discuss the usage of `grep ("ia", state.name)`.

c) Discuss the output of `objects (pos = grep("stats", search()))`.

d) Use paste to create variable names: var1, var2, …, var100.

e) Repeat (d) to create variable names: var_1, var_2, …, var_100.

f) Discuss the output of

```
substring (paste(letters,collapse=""),
  1:nchar(paste(letters,collapse="")),
  1:nchar(paste(letters,collapse="")))
```

g) Obtain a copy of the second paragraph in the Preface of this book in the R commands window. Use this copy to calculate the number of words as well as the total number of characters (including spaces between words) in the passage.

We are going to use several of the functions in Table 3.6.6 to perform this task in steps. First open the Preface in MS Word and then copy the second paragraph starting with 'Central' to the clipboard. Proceed as follows in R:

```
> TextPar <- scan(file="clipboard", what="")
```

To obtain a vector containing each of the words as a separate element.

```
> TextPar <- paste(TextPar,collapse = " ")
```

To convert `TextPar` into a vector containing one element consisting of all the words concatenated and separated by spaces into a single character string. Add the correct line breaks (`"\n"`) in `TextPar` using e.g. `fix()`.

```
> TextPar <- strsplit(x = TextPar, split = '\n')
> mode(TextPar)
[1] "list"

> mode(unlist(TextPar))
[1] "character"

> TextPar <- unlist(TextPar)
```

To change TextPar into a character vector

```
> nchar(TextPar)
> length(TextPar)
```

3.7 Differentiation and integration

3.7.1 Symbolic differentiation

Study the help files of `D()` and `derive()`.

3.7.2 Integration

Study the help file of `integrate()`.

3.8 Exercise

1. It is known from elementary statistics that approximately 68% of data from a normal distribution with a mean of zero and a standard deviation of unity will have an absolute value less than unity. Use the `sum()` and `rnorm()` functions to find the proportion of n random $normal\ (0;\ 1)$ variables whose absolute value is less than 1.0. Repeat with different values for n to investigate how widely the results vary.

2. Define: *conditional inverse* and *generalized (Moore-Penrose) inverse* for matrix $\mathbf{X}$: $p \times q$ and make provision for $p = q$, $p > q$ and $p < q$. First, show how the svd of $\mathbf{X}$ can be used to obtain a conditional inverse, $\mathbf{X}^c$ for $\mathbf{X}$. Now use the above information to write an R function for calculating $\mathbf{X}^c$ for any given $\mathbf{X}$. The function must provide a test to check if the calculated conditional inverse is indeed a conditional inverse. Illustrate the usage of your function.

3. Give the necessary instructions to:

   (i) read into R an external text data file consisting of 10 sample observations with each consisting of one character variable and two numerical variables.

   (ii) read into R a *large* external text data file consisting of 50 numerical variables but unknown number of records. Each record in this data file takes up 5 lines. The variables in the R object must have the names `X1`, …, `X50`.

4. Discuss the meaning of the following R instructions:
   1. `y <- x[!is.na(x)]`
   2. `z <- (x + y)[!is.na(x) & x >0]`
   3. `a <- x[-(1:5)]`
   4. `x[is.na(x)] <- 0`

4 Introducing traditional R graphics

A basic knowledge of R graphics is needed before directing attention to the art of writing programs (functions) in R. Therefore In this chapter a brief overview is given of the basics of traditional R graphics. In a later chapter, after studying the principles of R programming, a second round of R graphics will follow.

4.1 General

Study the graphical parameters by requesting

```
> ?par ↲
```

In Figure 4.1.1 the main components of a graph window are illustrated. Study this figure in detail. The ***Plot Region*** together with the ***margins*** is called the ***Figure Region***.

a) What is the difference between high-level and low-level plotting instructions?

b) Take note especially how the functions `windows()`, `win.graph()` or `x11()` are used as well as the different options available for these functions.

c) The instruction `dev.new()` allows opening a new graph window in a platform-independent way

d) In this chapter some high-level plotting instructions are studied. Each of these instructions results in a (new) graph window with a complete graph drawn. The command `graphics.off()` deletes all open graphic devices.

e) Study the use of `par()`, `par(mfrow = )` and `par(mfcol = )`. Study the use of `par(new=TRUE)` to plot more than one figure on the same set of axes.

f) Study how the functions `graphics.off()` and `dev.off()` work.

4.2 High-level plotting instructions

a) Construct a barplot of the illiteracy of the states according to the `areagrp` (as defined in Chapter 3) states in the `state.x77` dataframe. *Hint*: The function `tapply()` operates on a vector given as its first argument. Its second argument groups the first argument into groups so that the function given in its third argument can be applied to each of these groups. Study the following command:

```
> barplot(tapply(state.x77[, "Illiteracy"],areagrp,mean),
names=levels(areagrp), ylab = "Illiteracy", xlab = "Area of
State", main = "Barplot of Mean Illiteracy")
```

b) Construct for the `state.x77` data set box plots of illiteracy broken down by the income of the states. First use `cut()` to form three categories of state income:

```
> state.income <- cut(state.x77[ , "Income"], c(0,4000,5000,Inf
),labels=c("$4000 or less", "$4001-$5000", "more than $5001"))
```

Then use `boxplot()` together with `split()` to produce the desired graph:

```
> boxplot(split(state.x77[ , "Income"], state.income))
```

Add labels for the axes as well as a title for the figure.

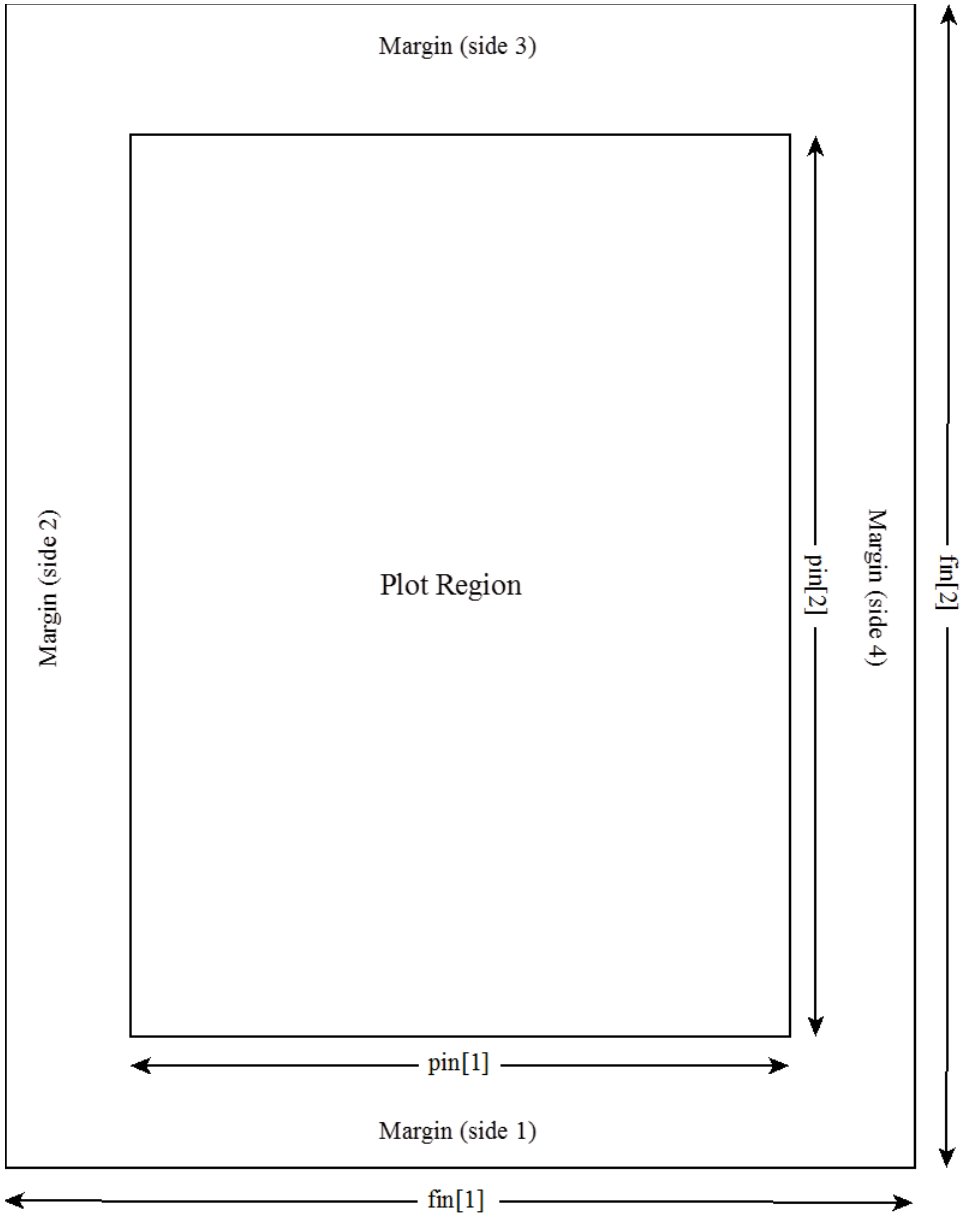


Figure 4.1.1: The main components of a graph window and the parameters for controlling their sizes. The parameter `mai` is a numerical vector of the form `c(bottom, left, top, right)` specifying the margins in inches while the parameter `mar` has a similar form specifying the respective margins as the number of lines. The default of `mar` is `c(5, 4, 4, 2) + 0.1`.

c) Repeat the previous example but use argument `notch = TRUE`.

d) Repeat the previous example but use function `my.boxplot()` (available using DropBox link https://drive.google.com/open?id=1k8QfGGT9SkViCcLfjBObj4fhtZ1kwI71) which contains some enhancements made to the built-in `boxplot()` function. Note `my.boxplot ()` requires its first argument to be a dataframe with two columns. The second column is a grouping vector. Therefore the following set of instructions can be used:

```
> split.out <- split(state.x77[ , "Income"], state.income)
> names(split.out)
[1] "$4000 or less" "$4001-$5000" "more than $5001"
```

```
> split.groupvec <- rep(names(split.out), c(length(split.
out[[1]]),length(split.out[[2]]), length(split.out[[3]])))

> my.boxplot(data=data.frame(unlist(split.out), split.
groupvec), notch=TRUE)
```

Study the above commands carefully.

e) Attach the package `akima`. What is the usage of the function `interp()`?
Discuss by constructing the following contour plot:

```
> contour(interp(state.center$x, state.center$y, state.
x77[,"Frost"]))
```

f) What is a *coplot*? Discuss after giving the following instruction and referring to the role of the tilde (~) operator.

```
> coplot(state.x77[,"Illiteracy"] ~ state.x77[,"Area"] | state.
x77[,"Income"])
```

g) A *dotchart* is constructed with function `dotchart()`. First some preparations are necessary:

```
> incgroup <- cut(state.x77[,"Income"], 3, labels = c("LowInc
","MediumInc","HighInc"))
> lifgroup <- cut(state.x77[,"Life Exp"], 2, labels =
c("LowExp","HighExp"))
> table.out <- tapply(state.x77 [,"Income"],
list(lifgroup,incgroup), mean)

> table.out
           Low Medium High
LowExp 3640.917 4698.417 5807
HighExp 4039.600 4697.667 5348

> dotchart (table.out, levels(factor(col(table.out),
labels=levels(incgroup)))[col(table.out)], factor(row(table.
out), labels=levels(lifgroup)))
```

Complete the graph by adding a label to the x-axis and a heading for the graph.

h) Use function `faces()` available in package `aplpack` to construct Chernoff faces for the Western states in the data set `state.x77`. *Hint*: The Western states appear in rows 3, 5, 12, 26, 28, 37, 44, 47 and 50. Explain what is represented by each of the facial features. First set argument `face.type = 0` and then `face.type = 1`.

i) Obtain a histogram of the life expectancy in the states of `state.x77`.

j) Execute the command

```
> pairs (state.x77)
```

Interpret the graph.

k) Three-dimensional graphs are constructed with function `persp()`.

```
> pts <- seq(from=-pi, to=pi, len=20)
> z <- outer(X=pts,Y=pts, function(x,y) sin(x)*cos(y))
> persp(x=pts, y=pts, z, theta=10, phi=60, ticktype = 'detailed')
```

Discuss the meaning of each of the above instructions. Experiment with different values for arguments `theta` and `phi`.

l) Obtain a pie chart of the object `areagrp` defined in Chapter 3. *Hint*: function `table()` may be useful here.

m) A cluster plot (dendrogram) can be constructed with function `hclust()` as follows:

```
> distmat.west <- dist (scale (state.x77
[c(3,5,12,26,28,37,44,47,50),]))
> plot(hclust(distmat.west), labels = rownames(state.x77)
[c(3,5,12,26,28,37,44,47,50)])
```

Interpret the above instructions and the resulting plot.

n) Use the function `plot()` to plot *sin* θ as theta varies from $-\pi$ to π.

o) Could you explain the different graphs resulting from the two calls to the `plot()` function above?

p) Obtain the empirical distribution function of variable `Life Exp` in the `state.x77` data set by using the functions `cut()`, `ecdf()` and `plot()`.

q) Check the normality of variable `Income` in the `state.x77` data set by using function `qqnorm()`.

r) Obtain a qqplot of the income of small states versus the income of large states in the data set `state.x77` where small and large are defined as below or above the median income, respectively.

```
> state.size <- cut(state.x77[,"Area"], c(0, median(state.
x77[,"Area"]), max(state.x77[,"Area"])))
> state.income <- split(state.x77[,"Income"], state.size)
> qqplot(state.income[[1]], state.income[[2]], xlab="Income for
small states", ylab="income for large states")
```

s) Use function `ts.plot()` to construct a time series plot of the `sunspots` data set.

4.3 Interactive communication with graphs

a) Study the help files of the functions `text()`, `identify()` and `locator()`.

b) Illustrate the usage of `identify()` on a scatterplot of variables `Illiteracy` and `Life Exp` in the `state.x77` data set:

```
> plot(x=state.x77[,'Life Exp'],y=state.x77[,'Income'])
```

To create the scatterplot, then call

```
> identify (x = state.x77[,'Life Exp'], y = state.x77[,'Income'],
seq (along = rownames(state.x77)), n = 5)
```

Notice the change in the cursor; the cursor changes to a cross when moved over the graph. However the cursor over a point to identify and click left mouse button. Repeat $n = 5$ times. Explain the result. Next, create the scatterplot once more and then call

```
> identify(x = state.x77[,'Life Exp'], y = state.x77[,'Income'],
labels = rownames(state.x77)
[seq (along = rownames(state.x77))] , n = 5)
# Explain what has happened.
```

c) Illustrate the usage of `locator()` by.

i. Joining 5 user defined points on a graph interactively with straight lines

```
> plot(x=state.x77[,'Life Exp'], y=state.x77[,'Income'])
> locator(5, type = "l")
```

Use mouse and select the five points on the graph

What happened on the graph?

What happened in the commands window?

ii. Writing text interactively at a specified position on an existing graph

```
> plot(x=state.x77[,'Life Exp'], y=state.x77[,'Income'])
> text(locator(n=1,type="n"), label = "State with the highest
income")
```

4.4 3D graphics: Package rgl

Write and execute the following function

```
rgl.example <- function (size=0.1, col="green", alpha.3d=0.6)
{ require(rgl)
 datmat <- matrix(rnorm(30),ncol=3)
 open3d()
 spheres3d(datmat,radius=size, color=col,alpha=alpha.3d)
 axes3d(col = "black")
 device.ID <- rgl.cur()
 answer <- readline("Save 3D graph as a .png file? Y/N\n")
 while(!(answer == "Y" | answer == "y" | answer == "N" |
       answer == "n"))
   answer <- readline("Save 3D
 graph as a .png file? Y/N \n")
 if (answer == "Y" | answer == "y")
 repeat
 { file.name <- readline("Provide file name including full
                          path NOT in quotes and SINGLE
                          back slashes! \n")
```

```
        file.name <- paste(file.name, ".png", sep = "")
        snapshot3d(file = file.name)
        rgl.set(device.ID)
        answer2 <- readline("Save another 3D graph as a .png
                            file? Y/N \n")
        if (answer2 == "Y" | answer2 == "y") next else break
      }
  else rgl.set(device.ID)
}
```

Study the above code and constructions in detail.

4.5 Exercise

1. Obtain a graph of a normal(100; 25) probability density function (p.d.f.)
2. Plot on the same set of axes
   i. a central beta(9; 5) p.d.f.;
   ii. a non-central beta(9; 5) p.d.f. with non-centrality parameter = 15 and
   iii. a non-central beta(9; 5) p.d.f. with non-centrality parameter = 40.

   Add a suitable legend to the plot.

3. Use `persp()` to obtain a graph of any user specified bivariate function. The challenge is that the function specification must appear as the main title of the graph. In order to address this problem we need information about the arguments of `persp()`:

```
> args(persp)
function (x, …)
NULL
This is not very helpful so we try
methods(persp)
[1] persp.default*
Non-visible functions are asterisked
args(persp.default)
Error in args(persp.default) : object 'persp.default' not found
```

The reason for this error message follows from the above as that `persp.default` is not visible. The immediate visibility of a function is regulated by a package builder through the package's namespace mechanism. Only object names that are exported are immediately visible; object names that are not exported are marked with an asterisk and are not visible. The functions `argsAnywhere()` and `getAnywhere()` are available to get information on asterisked object names:

```
> argsAnywhere(persp.default)
function (x = seq(0, 1, length.out = nrow(z)), y = seq(0, 1,
length.out = ncol(z)), z, xlim = range(x), ylim = range(y),
zlim = range(z, na.rm = TRUE), xlab = NULL, ylab = NULL,zlab =
NULL, main = NULL, sub = NULL, theta = 0, phi = 15,r = sqrt(3),
d = 1, scale = TRUE, expand = 1, col = "white",border = NULL,
ltheta = -135, lphi = 0, shade = NA, box = TRUE, axes = TRUE,
nticks = 5, ticktype = "simple", …)
NULL
```

We notice that we can make use of the argument `main` in a call to `persp()` to provide our perspective plot with a title. However, `main` accepts only character strings and not mathematical expressions. Furthermore, we have seen in the `persp()` example in section 4.2 that the values for the argument `z` are conveniently found by a call to `outer()` using its argument `FUN`. However `FUN` requires a function. So we need the means to convert expressions into character strings and vice versa to convert character strings into expressions.

The following pairs of functions allow these conversions to be made:

Character strings (" ") → expressions: `parse()` and `eval()`

Expressions (unquoted) → character strings (" "): `deparse()` and `substitute()`

```
> pts <- seq(from = -3, to = 3, len = 50)
> fun1 <- "2 * pi * exp( - (x^2 + y^2)/2)"
> fun2 <- parse(text = paste("function(x,y)", fun1))
```

Explain carefully what `parse()` is doing.

```
> zz <- outer(pts, pts, eval(fun2))
```

Explain carefully what `eval()` is doing.

```
> persp(x = pts, y = pts, z = zz, theta = 0, phi = 15,
ticktype = "detailed", main = paste("Persp plot of `",
fun2,"`",sep=""))
```

Explain carefully the role of paste()

4. (i) Use `persp()` to obtain a perspective plot of the `volcano` data.

   (ii) Obtain an RGL plot of the `volcano` data.

5 Subscripting

Vectorized arithmetic and subscripting are two cornerstones of R programming. Review section 4.2 for several examples where subscripting has been used. In this chapter subscripting is studied in detail. Specifically the following two related topics are studied:

- Extracting parts of an object by using ***subscripting***.
- The combination and rearranging of data within data structures like matrices, dataframes and lists.

5.1 Subscripting with vectors

The different types of subscripting with vectors are summarized in Table 5.1.1:

Type	Effect	Example
empty	Extract all values	`x[ ]`
integer, positive	Extract all values specified by the subscript	`x[c(2:5,8,12) ]`
integer, negative	Extract all values except those specified by the subscript	`x[-c(2:5,8,12) ]`
logical	Extract those values for which subscript is TRUE	`x[x > 5 ]`
character	Extract those values whose names attributes correspond to those specified by the subscript	`x[c("a","d") ]`

Table 5.1.1: Different types of subscripting vectors.

Logical subscripting provides a very powerful operation in R. A logical subscript is a vector of `TRUE`s and `FALSE`s that must be of the same length as the object being subscripted e.g.

```
> state.x77[ , "Area"] > 80000
```

gives a vector of `TRUE`s and `FALSE`s that can be used for selecting the rows of a matrix e.g.

```
> state.x77[state.x77[ , "Area"] > 80000 , "Income" ]
```

Select rows

Select column(s)

- What is the value of:

```
> xx[is.na(xx)]; > xx[!is.na(xx)]
```

- Study the following code:

```
> x <- c(10,15,12,NA,18,20)
> mean(x)
[1] NA
> mean(x[!is.na(x)])
[1] 15
> mean(na.omit(x))
[1] 15
```

- Logical subscripting allows finding the indices of those elements in a vector that meet a certain condition e.g.

```
> (1:length(rownames(state.x77)))[state.x77[ ,"Income"]>5000]
[1] 2 5 7 13 20 28 30 34
```

and to find the corresponding names of the states

```
> rownames(state.x77)[(1:length(rownames(state.x77)))
 [state.x77[ ,"Income"]>5000]]
 [1] "Alaska" "California" "Connecticut" "Illinois" [5] "Maryland"
"Nevada" "New Jersey" "North Dakota"
```

- In addition to extracting elements, the above subscripting operations can also be used to modify selected elements of a vector e.g. changing NA-values to zero:

```
> xx[is.na(xx)] <- 0
```

When the right-hand side of the assignment above is a scalar value, each of the selected values will be changed to the specified scalar value; if the right-hand side is a vector, the selecting values will be changed in order, ***recycling*** the values if more values were selected on the left-hand side than were available on the right-hand side.

5.2 Subscripting with matrices

Element and submatrix extraction of matrices are discussed below.

a) Revise the use of `matrix()`, `names()` , `dim()` and `dimnames()`.

b) A matrix in R is an ***array*** with two indices. Arrays of order two and higher can be constructed with the function `dim()` or `array()`.

Let, for example, ***a*** be a vector consisting of 150 elements. The instruction

```
dim(a) <- c(3,5,10)
```

or the instruction

```
a <- array(a,dim=c(3,5,10))
```

constructs a $3 \times 5 \times 10$ array.

- Matrices can therefore be formed as above, but the function `matrix()` is usually easier to use.
- The elements of a *2*-dimensional array can also be extracted using the one-index or two-index method as described below.

c) The subscripting methods described in §5.1 can also be applied to both the first or second dimension of a matrix where the first dimension refers to the rows and the second dimension to the columns of the matrix.

d) Note that the elements of a matrix can be referred to by the two-index method above or by a one index method. When the one index method is used it is assumed that the matrix has first been strung out ***column***-wise into a vector.

```
> testmat.a <- matrix( c(17,40,20,34,21,12,14,57,78,37,29,64),
  nrow = 4)
> testmat.a
     [,1] [,2] [,3]
[1,]   17   21   78
[2,]   40   12   37
[3,]   20   14   29
[4,]   34   57   64
> testmat.b <- matrix (c(17,40,20,34,21,12,14,57,78,37,29,64),
nrow = 4, byrow = TRUE)
> testmat.b
     [,1] [,2] [,3]
[1,]   17   40   20
[2,]   34   21   12
[3,]   14   57   78
[4,]   37   29   64
```

Comment on the difference between `testmat.a` and `testmat.b`

```
> testmat.a[2,3] # Two index matrix reference
[1] 37
> testmat.a[10] # One index matrix reference
[1] 37
```

e) Write a function to convert a one-index to a two-index matrix reference. Give an example of the usage of your function.

f) Write a function to convert a two-index to a one-index matrix reference. Give an example of the usage of your function.

g) Consider the following example to form submatrices:

```
> testmat <- matrix(1:50, nrow=10, byrow=TRUE)
```

```
> testmat[1:2,c(3,5)]
     [,1] [,2]
[1,]    3    5
[2,]    8   10
> testmat[1:2,3]
          [1] 3 8
> testmat[1:2,3,drop=FALSE]
     [,1]
[1,]    3
[2,]    8
```

h) Notice the difference between `testmat [1:2, 3]` and testmat `[1:2, 3, drop = FALSE]`. The first command results in the output to be given in the form of a vector while the optional `drop = FALSE` in the second command retains the matrix structure of the output. This distinction can have serious consequences when a procedure expects a matrix argument and not a vector.

i) Notice also that the output of both `testmat[1:2,3]` and `testmat[3, 1:2]` has a similar form: R makes no distinction between column vectors and row vectors; all one-dimensional collections of numbers are treated identically.

j) Apart from using vectors as subscripts to a matrix, a matrix can also be used as a subscript to a matrix. There are two cases:
(A) a numeric subscripting matrix and
(B) a logical subscripting matrix.

Case A

Here the subscripting numeric matrix must have exactly two columns: the first provide row indices and the second column indices.

1. If used on the right-hand side of an expression the result of a ***case A*** subscripting is a vector containing the values specified by the subscripting matrix.

2. If used on the left-hand side of an assignment a numeric matrix first selects those elements specified by its row and column indices; then these values are replaced one by one with the objects specified by the right-hand side of the assignment.

Here is an example of ***case A*** subscripting with the subscript matrix on the right-hand side of the assignment:

```
> xmat <- matrix(1:25, nrow=5)
> xmat
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

> superdiag.index <- matrix(c(1:4,2:5), ncol=2, byrow=FALSE)
> superdiag.values <- xmat[superdiag.index]
> superdiag.values
     [1]  6 12 18 24
```

Case A subscripting with the numeric subscript matrix on the left-hand side of the assignment:

```
> subscript.mat <- matrix (c(1:3, 1:3, rep(1,3), rep(2,3)), ncol=2)
> subscript.mat
```

```
      [,1] [,2]
[1,]    1    1
[2,]    2    1
[3,]    3    1
[4,]    1    2
[5,]    2    2
[6,]    3    2

> xx <- matrix(NA, nrow=3,ncol=2)
> xx

      [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
[3,]   NA   NA

> xx[subscript.mat] <- c(10,12,14,100,120,140)
> xx

      [,1] [,2]
[1,]   10   100
[2,]   12   120
[3,]   14   140
```

Case B

The logical subscripting matrix must be in size exactly similar to that matrix it is subscripting and will select those values corresponding to a `TRUE` in the subscripting matrix.
Case B with logical subscripting matrix at right-hand side of assignment:

```
> testmat
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
[5,]   21   22   23   24   25
[6,]   26   27   28   29   30
[7,]   31   32   33   34   35
[8,]   36   37   38   39   40
```

```
[9,]    41    42    43    44     45
[10,]   46    47    48    49     50

> aa <- testmat[testmat < 12]
> aa
[1] 1 6 11 2 7 3 8 4 9 5 10
```

Note that the selected elements are placed *column-wise* in a vector.

Case B with logical subscripting matrix at left-hand side of assignment:

```
> testmat[testmat < 12] <- 12
> testmat

      [,1] [,2] [,3] [,4] [,5]
[1,]    12   12   12   12   12
[2,]    12   12   12   12   12
[3,]    12   12   13   14   15
[4,]    16   17   18   19   20
[5,]    21   22   23   24   25
[6,]    26   27   28   29   30
```

```
[7,]    31    32    33   34    35
[8,]    36    37    38   39    40
[9,]    41    42    43   44    45
[10,]   46    47    48   49    50
```

In order to restrict assignment to a subset of a matrix two sets of subscripts are needed. See example below:

```
> testmat <- matrix(1:50, nrow=10, byrow=TRUE)
> testmat[, c(1,3)][testmat[,c(1,3)] <12] <- 12

> testmat
       [,1] [,2] [,3] [,4] [,5]
 [1,]   12    2    12     4    5
 [2,]   12    7    12     9   10
 [3,]   12   12   13    14   15
 [4,]   16   17   18    19   20
 [5,]   21   22   23    24   25
 [6,]   26   27   28    29   30
 [7,]   31   32   33    34   35
 [8,]   36   37   38    39   40
 [9,]   41   42   43    44   45
[10,]   46   47   48    49   50
```

Study the use of functions `row()` and `col()` in constructing logical matrices.

5.3 Extracting elements of lists

a) Note the use of `list()` to collect objects into a list while elements are extracted with `$`

- the function `names()`,
- the single square brackets `[ ]` and
- the double square brackets `[[ ]]`.

b) Study the following example carefully:

```
> my.list <- list(el1 = 1:5, el2 = c("a","b","c"), el3 =
matrix(1:16,ncol=4), el4 = c(12,17,23,9))

> my.list
$el1
```

```
[1] 1 2 3 4 5
$el2
[1] "a" "b" "c"
$el3
      [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
$el4
[1] 12 17 23 9

> my.list$el2
 [1] "a" "b" "c"

> mode(my.list$el2)
 [1] "character"

> my.list[el2]
 Error: object 'el2' not found

> my.list["el2"]
 $el2
 [1] "a" "b" "c"

> mode(my.list["el2"])
 [1] "list"

> my.list[["el2"]]
 [1] "a" "b" "c"

> mode(my.list[["el2"]])
 [1] "character"
```

Note: The above example shows that using the single pair of square brackets for subscripting a list always result in a list object to be returned. This is often the cause of an error message. See the example below.

```
> my.list[1]
 $el1
 [1] 1 2 3 4 5

> mode(my.list[1])
 [1] "list"

> my.list[[1]]
 [1] 1 2 3 4 5

> mode(my.list[[1]])
 [1] "numeric"

> my.list[3][2,4]
 Error in my.list[3][2, 4] : incorrect number of

 dimensions

> my.list[[3]][2,4]
 [1] 14

> my.list$el3[2,4]
 [1] 14

> mean(my.list[4])
[1] NA
Warning message:
In mean.default(my.list[4]) :
 argument is not numeric or logical: returning NA

> mean(my.list[[4]])
[1] 15.25

> mean(my.list$el4)
[1] 15.25
```

Explain the differences and similarities between the symbols `[ ]`, `[[ ]]` and `$` when subscripting lists.

5.4 Extracting elements from dataframes

a) Note the use of `data.frame()` for creating dataframes. A dataframe has a rectangular structure similar to a matrix but differs from a matrix in that its columns are not restricted to contain the same type of data. Each of its columns must contain the same sort of data but some columns can be numerical while others are factors for example.

b) Explain the difference between the objects created by the following two instructions:

```
> my.matrix <- matrix(c(17,40,20,34,21,12,14,57,78,37,29,64),
nrow = 4, ncol = 3)

> my.dataframe <-
  data.frame(c(17,40,20,34,21,12,14,57,78,37,29,64),
  nrow=4,ncol=3)
```

c) Note the following

```
> class(my.matrix)
[1] "matrix"

> class(my.dataframe)
[1] "data.frame"
```

```
> is.list(my.dataframe)
[1] TRUE

> mode(my.matrix)
[1] "numeric"

> mode(my.dataframe)
[1] "list"
```

d) A sample of the behaviour of dataframes

```
> my.dataframe.2 <- data.frame(C1 =
c('a','b','c' ,'d'), C2 = c(5,9,23,17),
C3 = c(TRUE,TRUE,FALSE,TRUE))
```

Notice the consequence of the argument `stringsAsFactors` in the above function call.

```
> my.dataframe.2
  C1 C2   C3
1 a   5   TRUE
2 b   9   TRUE
3 c 23 FALSE
4 d 17   TRUE

> my.dataframe.2[ ,1:2]
  C1   C2
1  a   5
2  b   9
3  c 23
4  d 17
```

In the above the dataframe behaves like a matrix.

```
> my.dataframe.2$C1
[1] a b c d
Levels: a b c d
```

Now the dataframe behaves like a list.

```
> as.matrix(my.dataframe.2)
     C1 C2    C3
[1,] "a" " 5" " TRUE"
[2,] "b" " 9" " TRUE"
[3,] "c" "23" "FALSE"
[4,] "d" "17" " TRUE"
```

Explain what has happened above.

e) The above examples show that a dataframe can be considered as a cross between a matrix and a list. Therefore subscripting of dataframes generally can be performed using the basic techniques available for matrices and lists.

f) An alternative technique is to extract the elements of a list by using the functions `attach()` and `names()`. This technique is especially of importance in statistical modelling. What is a potential danger of this technique when attaching dataframes? This danger can be avoided by using `with()`. Is this also true when modelling is performed?

g) Review section 2.3. Study the help file of the function `with()`. What important usage has `with()`?

5.5 Combining vectors, matrices, lists and dataframes

a) What is the result of the command
`> my.list <- vector("list", k)`?

b) Recall the function `c()` for creating vectors. When `c()` is used to combine a numeric vector and a character vector the result is a vector of mode "character". Similarly, using `c()` to combine a vector with a list results in a list.

c) If `list()` is used to combine two or more vectors or lists the result is a list of all the objects.

d) The function `unlist()` can be used to convert all the elements of a list into a single vector.

```
unlist(my.list)
 el11  el12   el13   el14  el15   el21  el22   el23
  "1"   "2"    "3"    "4"   "5"    "a"   "b"    "c"
 el31  el32   el33   el34  el35   el36  el37   el38
 "1"   "2"    "3"    "4"   "5"    "6"   "7"    "8"
 el39  el310  el311  el312 el313  el314  el315  el316
 "9"   "10"   "11"   "12"  "13"   "14"  "15"   "16"
 el41  el42   el43   el44
 "12"  "17"   "23"   "9"
```

Explain the above output.

e) Review the functions
`cbind()`, `rbind()`, `append()`, `data.frame()`, `dim()`, `dimnames()`, `names()`, `colnames()`, `rownames()`, `nrow()` and `ncol()`.

5.6 Rearranging the elements in a matrix

Study the usage of the functions `matrix()`, `t()` and `diag()`. These functions are useful to form submatrices of a matrix or to rearrange matrix elements. Note again the argument `byrow` of `matrix()`.

5.7 Exercise

1. Write an R function to check if a given matrix is symmetric.
2. Write an R function to extract
   i. the row(s) and
   ii. the columns

   containing the maximum value in the matrix. Note that provision must be made that the maximum value can occur in more than one row (column). Furthermore, both the indices and actual values of the rows (columns) must be returned. Illustrate the usage of your function with a suitable example.
3. Describe the variables in the built-in data set `LifeCycleSavings`. Is this data set in the form of a matrix or a dataframe?
4. Use subscripting to find the largest proportion of over 75 in those countries with a dpi of less than 1000 in the `LifeCycleSavings` data set. Determine also the country(ies) having this pop75 value.
5. Consider the `LifeCycleSavings` data set.
   i. Use subscripting to find the mean aggregate savings for countries with a percentage of the population younger than 15 at least 10 times the percentage of the population over 75.
   ii. Find also the mean aggregate savings for countries where the above ratio is less than 10.
   iii. Use function `t.test()` to test if mean aggregate savings are different for the above two groups.
   iv. Use notched box plots for an approximate test.
   v. First, study carefully the output obtained in (iii) and (iv). Then interpret/discuss this output in detail.
6. Consider the `state.x77` data set and the variable `state.region`. Find the state with the minimum income in each of the regions defined in `state.region`.

6 Revision tasks

In general the purpose of writing a program in R is to address some practical problem directly or indirectly. To prepare the student for seriously writing R functions (programs) this chapter consists of a mixture of revision tasks. While some of these tasks are straight forward others need more thought and preparation before starting with the writing of R code. In Section 6.1 some guidelines are considered for writing R code to address a practical problem.

6.1 Guidelines for problem solving by writing R code

a) Make sure the problem is clearly understood. You cannot write good code for something that is not correctly grasped.

b) Break complex problems into simpler components. Formulate these simpler components in terms of specific questions to be answered.

c) Write dedicated code for answering the specific questions in (b).

d) Think in terms of the way R operates e.g. vectorized arithmetic, recycling principle, operating on objects as wholes/units, subscripting, R data structures…

e) Spend time to prepare your data.

f) Ask yourself the question what information do you need before attempting to write code for coming up with an answer. Then, what facilities are provided in R to get the necessary information and once the information is available what manipulations are needed to code useful output.

g) Do not neglect the debugging/optimizing phase of code that succeeds in providing a first round answer.

6.2 Exercise

1. Use R to obtain a five-point summary of the variable `dpi` in the `LifeCycleSavings` data set. Illustrate the difference between the working of `fivenum()` and `quantile()`. *Hint*: See `boxplot.stats()` for the definition of *hinges*.

2. Display the pdf of a *normal* (100, 15) distribution graphically. The area under the density bounded by the 70th and 90th percentiles must appear in red.

3. (i) Use R to obtain graphical representations of the pdf as well as the cdf of an F(10,15) and an F(15,10) stochastic variable. These graphs must be on one graph window and be supplied with suitable titles. Furthermore, they must be line graphs that contain no other plotting characters except lines.

   (ii) Obtain representations as line graphs of the inverses of the above cdfs on a single separate graph page.

4. First set the seed to 172389 and then generate a random sample of size 500 from a *normal* (100, 20) distribution. Give the necessary R instructions to determine the class frequencies in the class intervals "Smaller than 50", "50 to 75⁻", "75 to 90⁻", "90 to 100", "100⁺ to 110", "Larger than 110".

5. Generate a random sample of size 80 from a bivariate normal distribution with mean vector (50, 100). The variances of the two variables are 900 and 2500 respectively with a correlation 0.90. Store the sample in an R matrix object and obtain a scatterplot in the form of

   i. a point diagram and

   ii. a line graph of the sample.

6. Define the harmonic mean for a vector of observations. What conditions must be satisfied by the observations?

   i. Write your own function for calculating a harmonic mean and use it to calculate the harmonic mean of variable `dpi` in the `LifeCycleSavings` data set.

   ii. Calculate the ordinary mean of variable `dpi` in the `LifeCycleSavings` data set. Compare the answer with the answer in (a). Which answer would you use in practice? Motivate.

7. Fisher's linear discriminant function in the case of two groups is defined as follows:

a) $LDF = (\bar{\boldsymbol{x}}_1 - \bar{\boldsymbol{x}}_2)'\mathbf{S}^{-1}\boldsymbol{x}$ where $\mathbf{S} = [(n_1 - 1)\mathbf{S}_1 + (n_2 - 1)\mathbf{S}_2]/(n_1 + n_2 - 2)$ with $\bar{\mathbf{x}}_i$ and $\mathbf{S}_i$ the vector of means and the covariance matrix of the ith group (sample), respectively.

b) The corresponding classification function is written as $CF = (\bar{\boldsymbol{x}}_1 - \bar{\boldsymbol{x}}_2)'\mathbf{S}^{-1}\boldsymbol{x} - \frac{1}{2}(\bar{\boldsymbol{x}}_1 - \bar{\boldsymbol{x}}_2)'\mathbf{S}^{-1}(\bar{\boldsymbol{x}}_1 + \bar{\boldsymbol{x}}_2)$.
In agreement with section 6.1 make sure what an LDF and a CF entail.
The `company.10var` data set consists of financial data in the form of 10 ratio numbers of a sample of successful and unsuccessful companies. The first column of the data matrix is a grouping variable that indicates whether the company concerned was successful or unsuccessful. The data set is available as an R object to be sourced via the link
https://drive.google.com/open?id=1Yn_PZ5Jy1BvnYRRVDP_83M8hGidd3FB6

   i. Use function `source()` to obtain a copy of the `company.10var` data set as an R object.

   ii. Obtain the covariance matrix of (a) the successful as well as (b) the unsuccessful companies.

   iii. Obtain the vector of means of (a) the successful and (b) the unsuccessful companies.

   iv. Use standard R functions operating on matrices to write a function or code that calculates the discriminant coefficients for the given linear discriminant function. Next, adapt this function to return

      a) the linear discriminant function and

      b) the classification function for the `companies.10var` data set.

8. Consider the matrix $\mathbf{A}$: $n \times m$.
   What is understood by the column space $\boldsymbol{V}(\mathbf{A})$ and the orthogonal complement $\boldsymbol{V}^{\perp}(\mathbf{A})$? The R function `svd()` can be used to obtain an orthogonal basis for $\boldsymbol{V}(\mathbf{A})$ when the rank of $\boldsymbol{A}$ is k. We also want to determine an orthogonal basis for $\boldsymbol{V}^{\perp}(\mathbf{A})$. How can the function `svd()` be used to simultaneously find a basis for $\boldsymbol{V}(\mathbf{A})$ and for $\boldsymbol{V}^{\perp}(\mathbf{A})$?

The above propositions can be proved as follows: Assume that $\boldsymbol{V}^{\perp}(\mathbf{A})$. and that an orthonormal basis for $\boldsymbol{V}(\mathbf{A})$ as well as for $\boldsymbol{V}^{\perp}(\mathbf{A})$ must be found. Append $n - m$ zero vectors of size n to the matrix **A.** Write $\mathbf{A}^0$ for the appended matrix and perform the function `svd()` on $\mathbf{A}^0$. It follows that $\mathbf{A}^0 = \mathbf{UDV}'$ so that $\mathbf{A}^0\boldsymbol{V} = \boldsymbol{UD}$, i.e
$[\mathbf{A}^0\boldsymbol{v}_{(1)} \quad \mathbf{A}^0\boldsymbol{v}_{(2)} \quad \ldots \quad \mathbf{A}^0\boldsymbol{v}_{(n)}] = [d_1\boldsymbol{u}_{(1)} \quad \ldots \quad d_k\boldsymbol{u}_{(k)} \quad \mathbf{0} \quad \ldots \quad \mathbf{0}]$. Now
$\mathbf{A}^0\boldsymbol{v}_{(i)} \in V(\mathbf{A}^0) = V(\mathbf{A})$. (*Motivate in detail.*) It follows that $\boldsymbol{u}_{(i)} \in V(\mathbf{A}), i = 1, 2, \ldots, k.$ (*Motivate in detail.*) Therefore the columns of **U** that correspond to the non-zero *d*s form an orthonormal basis for $\boldsymbol{V}(\mathbf{A})$ while the columns of **U** that correspond to the zero *d*s form an orthonormal basis for the orthogonal complement of $\boldsymbol{V}(\mathbf{A})$. Motivate the last statement in detail.

9. Based on the results in (8) above, write an R function that *rank* $\boldsymbol{V}(\mathbf{A})$, an orthogonal basis for $\boldsymbol{V}(\mathbf{A})$ and an orthogonal basis for $\boldsymbol{V}^{\perp}(\mathbf{A})$. Test your function on the matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 2 & 4 \\ 3 & 2 & 7 \\ -1 & -5 & 2 \\ 2 & 7 & -1 \end{bmatrix}$$

10. In many graphical displays whose purpose it is to represent distances in two dimensions, it is essential that the scales of the axes are geometrically accurate. This is called the *aspect ratio* of the graph and the R graphics parameter asp is used for controlling the aspect ratio of graphics in R. The default value of asp generally does not ensure that the scales of the horizontal and vertical axes are geometrically accurate. ***For ensuring geometrically accurate scales the setting asp = 1 must be explicitly specified*** e.g. `plot(x =, y =, asp = 1).`

We are going to investigate the effect of the aspect ratio on graphs by writing our own function for drawing a circle. In agreement with section 6.1 we will start our project by reviewing some basic concepts regarding coordinates for graphical purposes. Figure 6.2.1 summarizes how to reference a point in geometric space by using (a) Cartesian coordinates and (b) polar coordinates.

i. Consider the following function for drawing a circle with a specified radius and centred at the origin:

```
my.circle <- function (r=1, xrange=-2:2, yrange=-2:2)
{ plot (x=xrange, y=yrange, type='n', xlab='', ylab='',
     xaxt='n', yaxt='n')
 theta <- seq(from=0, to = 2 * pi, by = 0.01)
 # Notice the use of radians.
 lines (x = r*cos(theta), y = r*sin(theta))
 abline(h = 0)
 abline(v = 0)
}
```

Run the above function and consider the graph window. Increase and decrease the size of the graph window by dragging its edges. Does the figure look like a circle?

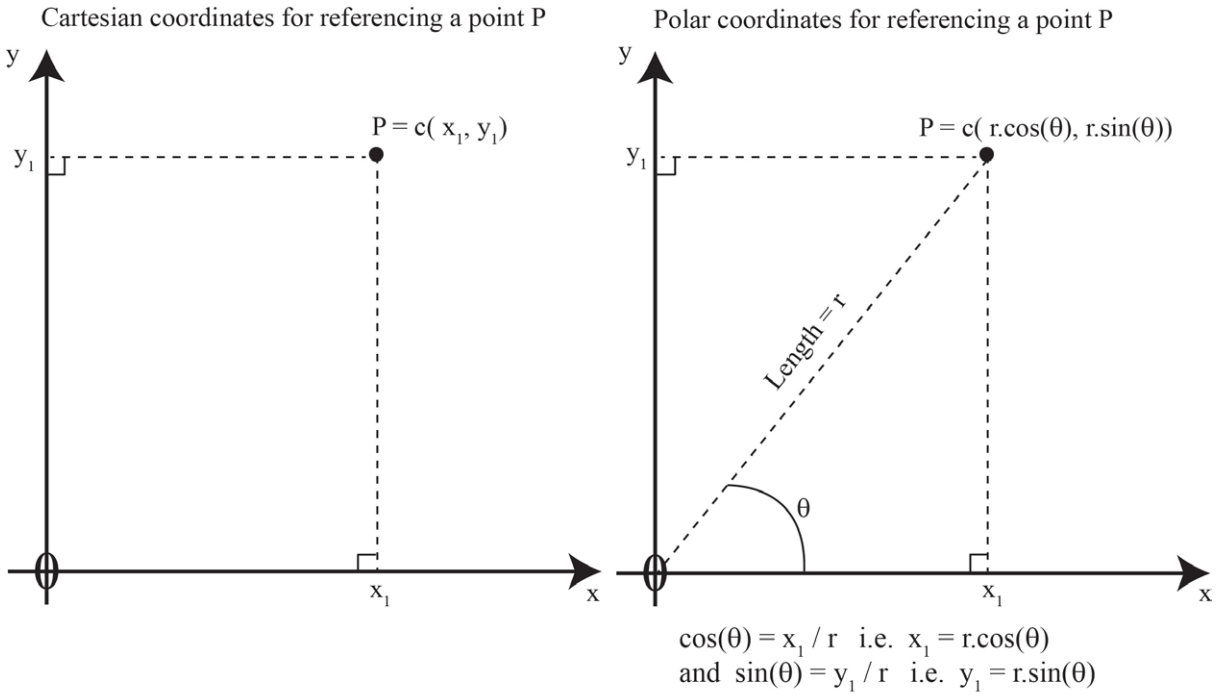


Figure 6.2.1: Cartesian and polar coordinates for referencing a point on a graph.

ii. Next, add the argument asp = 1 to the call to plot in my.circle. Run the changed function; change size of graph window. What happens?

iii. What changes are necessary for producing a circle centred at any point in a geometrical space? Make the necessary changes in `my.circle()` for constructing a circle centred at any user specified point on a graph.

11. What is understood by a p-dimensional ellipsoid?

i. Give a mathematical expression in matrix notation that describes an ellipsoid in p dimensions.

ii. Describe the axes of the ellipsoid in terms of eigenvalues and eigenvectors.

iii. Let $p = 2$. Simplify the expression for the ellipse concerned in terms of scalar quantities.

iv. Use `plot()` and write an R-function to draw an ellipse. Make provision for the centre point to be at (0, 0) as well as at an arbitrary (x_1, x_2) point; for no correlation between the two variables as well as for positive and negative correlation; for one or both axes to coincide with the coordinate axes.

v. Use your function written in (iv) to illustrate differences between `plot`(using the default value of argument `asp` and `plot` with `asp=1`.

12. Consider the following game. You are given a computer screen containing a rectangle filled at random with evenly spaced letters. Repetitions of the same letter are allowed. The challenge is to select sequentially the first n letters of the alphabet as quickly as possible. You must read each line from left to right and from top to bottom. Going backwards is not allowed. The time to complete the task is taken as well as if the rules have been obeyed. Program an R version of this game.

7 Writing functions in R

Although we have already written various functions in R, in this chapter the writing of R functions will be approached systematically.

7.1 General

A good way to learn about functions or to write a new function is to look at existing ones. As an example consider that we would like to write a function to implement a novel plotting procedure. So we start by taking a look at the existing `plot` function.

```
> plot ↲
function (x, y, …)
UseMethod("plot")
<bytecode: 0x000000000ee0c900>
<environment: namespace:graphics>
```

This is not very helpful so we give the instruction:

```
> methods(plot) ↲
 [1] plot.acf*              plot.data.frame*   plot.decomposed.ts*
 [4] plot.default           plot.dendrogram*   plot.density
 [7] plot.ecdf              plot.factor*       plot.formula*
[10] plot.function          plot.hclust*       plot.histogram*
[13] plot.HoltWinters*      plot.isoreg*       plot.lm
[16] plot.medpolish*        plot.mlm plot.     ppr*
[19] plot.prcomp*           plot.princomp*     plot.profile.nls*
[22] plot.spec              plot.stepfun       plot.stl*
[25] plot.table*            plot.ts            plot.tskernel*
[28] plot.TukeyHSD
 Non-visible functions are asterisked
```

If we decide to take a look at `plot.default` we can do so by

```
> plot.default ↲
function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL, log = "",
main = NULL, sub = NULL, xlab = NULL, ylab = NULL, ann = par("ann"),
axes = TRUE, frame.plot = axes, panel.first = NULL, panel.last = NULL,
asp = NA, …)
{
Lines of code are visible
}
```

Since our new plotting method is aimed at categorical data we decide rather to take a look at `plot.factor`. But this is an asterisked function and hence is not visible:

```
> plot.factor
Error: object 'plot.factor' not found
```

Asterisked functions can be inspected using the following method:

```
> getAnywhere(plot.factor)
A single object matching 'plot.factor' was found
It was found in the following places
 registered S3 method for plot from namespace graphics
 namespace:graphics
with value
function (x, y, legend.text = NULL, …)
{
      if   (missing(y) || is.factor(y)) {
            dargs <- list(…)
            axisnames <- if (!is.null(dargs$axes))
                  dargs$axes
```

```
            else if (!is.null(dargs$xaxt))
                  dargs$xaxt != "n"
            else TRUE
      }
      if   (missing(y)) {
            barplot(table(x), axisnames = axisnames, …)
      }
      else if (is.factor(y)) {
            if (is.null(legend.text))
                  spineplot(x, y, …)
            else {
                  args <- c(list(x = x, y = y), list(…))
                  args$yaxlabels <- legend.text
                  do.call("spineplot", args)
      }
      }
      else if (is.numeric(y))
            boxplot(y ~ x, …)
      else NextMethod("plot")
}
```

a) How are default values assigned to arguments of functions?

b) What is the default behaviour of `plot.factor()`?

c) What tasks can be achieved with `pmatch()` and what is understood by ***partial matching***? What will happen if `plot.factor()` is called with (i) `legend.text = 'AA=Agecat'`; (ii) `leg = 'AA=Agecat'`? Explain.

d) Discuss the usage of `missing()`.

e) Give an example of the usage of the function `stop(message= " ")`.

f) Give an example of the usage of the function `warning(message= " ")`.

g) What is the usage of the function `warnings()`?

h) Why can functions be called without specifying any arguments e.g. `q()`?

i) If the body of a function consists only of a single instruction it is not necessary to enclose it with braces.

j) The convention is to use the last evaluated statement as a function's return value. If several objects are to be returned gather them in a list.

k) The function `return()` with a single object or a ***list*** of objects is useful to interrupt a function at some intermediate stage and return an object or a list of objects at that particular stage. This is usually done when a function is under development.

l) Sometimes there is no meaningful value to return e.g. when a function is written primarily to produce some plot. In cases like this the function `invisible()` can be used as the last statement of the function. As an example of the usage of `invisible()` give the following instructions:

```
> boxplot(rnorm(100), plot = TRUE)
> boxplot(rnorm(100), plot = FALSE)
```

Now look at the end of function `boxplot.default()` to see how `invisible()` has been implemented.

m) Libraries (packages) of R functions. Attaching and detaching libraries to the search path. (Revise Chapter 1)

n) Creating a new function using scripts or `fix()`.(Revise Chapter 1)

o) Editing an existing function using scripts or `fix()`.(Revise Chapter 1)

p) Note that when writing a function a line can be interrupted at any place and be continued on a next line. ***Warning: Be careful not to put the break point where it marks the completion of an executable statement.*** Explain.

7.2 Writing a new function

Determining the indices of elements in a vector or matrix that meet a certain condition: the function `where()`

a) Write the following function using the command
```
> fix(where)
function(x, cond)
{ # Argument cond must evaluate to a logical value
    if(!is.matrix(x))
     seq(along = x)[cond]
    else matrix(c(row(x)[cond], col(x)[cond]), ncol = 2)
}
```

b) Inspect the *airquality* data set using the command `str(airquality)`.

c) Use the `where()` function to find the indices of (i) the NAs, (ii) the maximum value and (iii) the minimum value in the airquality data set.

d) Repeat (c) using the built-in function `which()`.

7.3 Checking for object name clashes

a) What happens if an R object is given the same name as an existing object?

b) Discuss the usages of the functions `apropos()`, `conflicts()`, `find()` and `match()` for the naming of objects.

c) Remember that when a function is called the R evaluator first looks in the global environment for a function with this name and subsequently in each of the attached packages or date bases in the order shown by `search()`. The evaluator generally stops searching when the name is found for the first time. If two attached packages have functions with the same name one of them will ***mask*** the object in the other. For example, the function `gam()` exists in two packages, `gam` and `mgcv`. If both were attached the command `find("gam")` will return
`[1] "package:gam" "package:mgcv".`

d) The operator `::` can be used to access the intended version of `gam()` by using the call `mgcv::gam()` or `gam::gam()`.

e) When writing R packages the ***namespace*** of the package provides another mechanism for ensuring that the correct version of a function is used. Note in this regard that the operator `:::` can be used to access objects that are not exported.

7.4 Returning multiple values

Exercise 7.4
Write an R function that returns the mean, median, variance, minimum, maximum and coefficient of variation of a numeric vector of sample data. ***The different components must be accessible by name.*** Test your function with the value of `rnorm(1000)`. *Hint*: Use the construct `list (mean = …, median = …, …)`.

7.5 Local variables and evaluation environments

a) Where is an object stored that is created by a script or `fix()`?

b) Where are local objects (objects that are created during the execution of a function) stored?

c) Explain how the evaluation environment works.

d) What is understood by the ***global environment***?

e) Study the R helpfile w.r.t. the operator <<−. When is it useful to use this operator? What are the dangers inherent to this operator?

f) What is understood by the ***scope*** of an expression or function?

The symbols which occur in the body of a function can be divided into three classes: ***formal parameters, local variables*** and ***free variables***. The formal parameters of a function are those appearing within the parentheses denoting the argument list of the function. Their values are determined by the process of ***binding*** the actual function arguments to the formal parameters. Local variables are created by the evaluation of expressions in the body of the functions. Variables which are neither formal parameters nor local variables are called free variables. Free variables become local variables when they are assigned to. Consider the following function definition.

```
fun <- function(datvec) {
            mean <- mean(datvec)
            print(mean)
            plot(datvec)
            plot(Traffic)
       }
```

In this function, `datvec` is a formal parameter, the object `mean` on the left-hand of the assignment symbol is a local variable (not to be confused with the function `mean()` on the right-hand side of the assignment symbol) while `Traffic` is a free variable. In R the free variable bindings are resolved by first looking in the environment in which the function was created. This is called ***lexical scope***.

If the following function call is made from the prompt in the working directory `fun(1:25)` the formal parameter `datvec` within the body of the function is assigned the value `1:25` (the ***actual argument***) and its mean is assigned to the local object `mean`. If the free parameter `Traffic` is found in the ***global environment*** or in a data base on the search path the required graph will be created else an error message will be sent to the console.

- Perform the above call.

7.6 Cleaning up

a) Study how the function `on.exit()` is used. This function can be used to reset options that are changed during an R-session back to their original values when the session is ended or a function terminates with an error message. It is also convenient for removal of temporary files.

b) Study the uses of the functions `.First()` and `.Last()`.

c) Write a function that automatically opens a graph window with a square plot region when an R-session is started.

7.7 Variable number of arguments: `argument ...`

a) Consider the following situation: You want to write a function for a complex task. At a particular stage a graph of some intermediate results is to be constructed. This requires the calling function to contain a call to the `hist` function. Here is an example of a chunk of code for executing this task:

```
complexfun <- function(datmat,colgraph)
     { datmat <- scale(datmat)
            # Several lines of complex code here
           hist(datmat, col = colgraph) }
```

A call like `complexfun(rnorm(1000), 'yellow')` can now be executed for the desired result. The problem is that the `hist` function has several arguments that you would like to be able to access by passing suitable actual values to them through the calling function `complexfun`. Instead of having to resort to provide a complete set of arguments in the argument list of `complexfun` R provides a neat way of addressing this situation: The argument ... which acts like any other formal argument except that it can represent a variable number of arguments. To see how the argument ... works change the above function to:

```
complexfun2 <- function(datmat, … )
 { datmat <- scale(datmat)
     # Several lines of complex code here
   hist(datmat, … ) }
```

Arguments represented by argument ... in the argument list of `hist` are passed to `hist` through the argument ... appearing in the arguments list of function `complexfun2` :

```
> complexfun2(datmat = rnorm(1000), col = 'yellow',
       probability = TRUE, xlim = c(-5,5))
```

b) Write a function that will retrieve the maximum length of any of an unspecified number of arguments of a specified mode. This is another illustration of the use of the ... argument:

```
maxlen <- function (mode.use="numeric", …)
 { my.list <- list(…)
    out <- 0
    for(x in my.list)
     if(mode(x) == mode.use) out <- max(out,length(x))
    out
 }
```

Note that the named argument must be specified as such in the function call:

```
> maxlen(1:10, 1:15, 1:3, letters)
> maxlen(mode.use="numeric", 1:10, 1:15, 1:3, letters)
> maxlen(1:10, 1:15, 1:3, letters, mode.use="character")
> maxlen(mode.use="character", 1:10, 1:15, 1:3, letters)
```

7.8 Retrieving names of arguments: Functions `deparse()` and `substitute()`

There are many practical situations requiring the conversion of mathematical expressions into character strings (text) or, conversely, requiring the conversion of text into mathematical expressions. The tools (functions) provided in R for achieving such conversions are summarized in Figure 7.8.1.

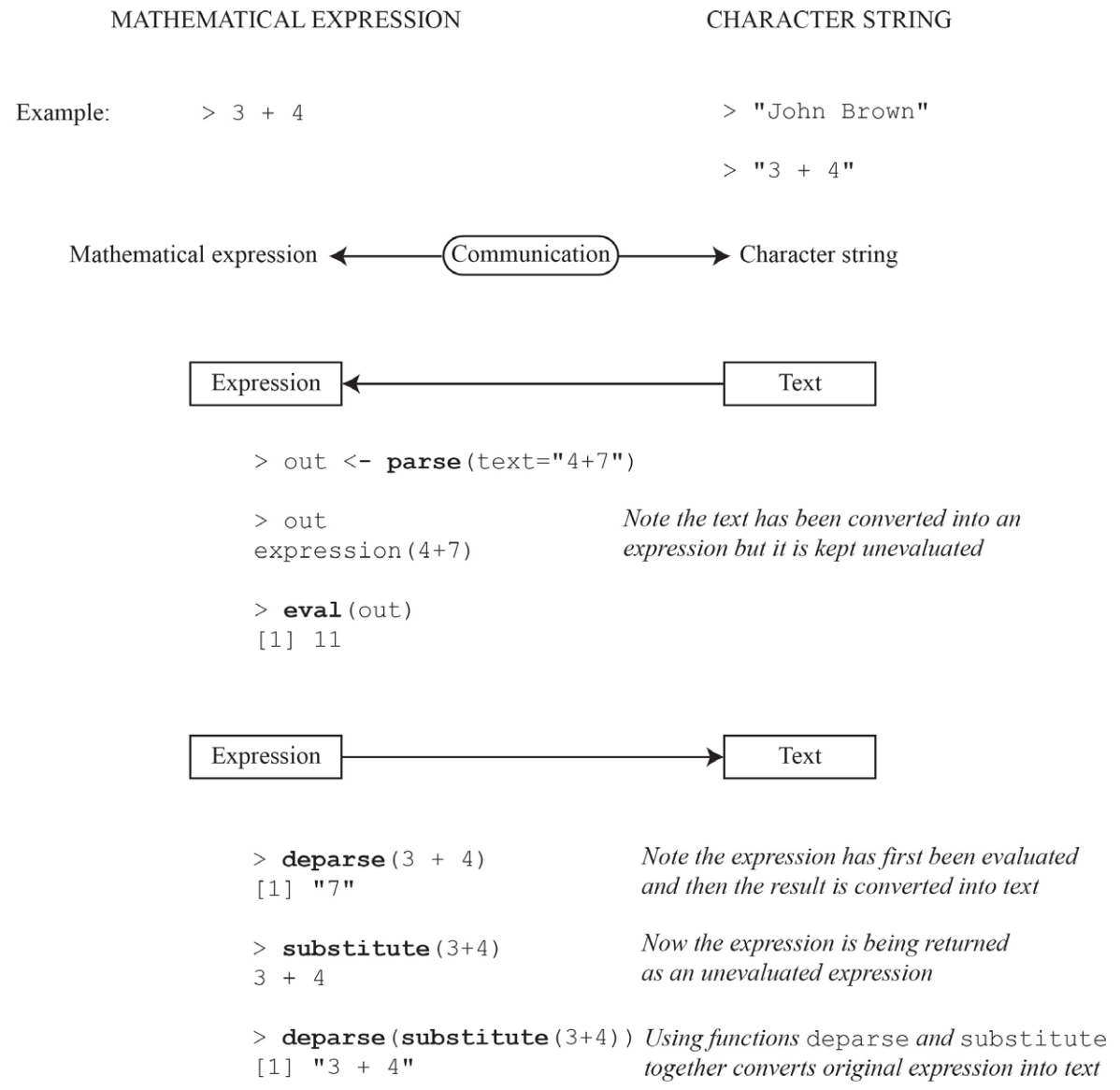


Figure 7.8.1: Converting text into mathematical expressions or mathematical expressions into text.

- Task: write an R function that will plot two vectors using as axis labels the names of the objects passed as arguments to the function as well as using these names in the main title of the graph.

It follows from Figure 7.8.1 that the function `substitute()` takes an expression as argument and returns it unevaluated. In order to evaluate the return value of `substitute()` the function `eval()` must be used. The function `deparse()` takes as argument an unevaluated expression and converts it into a character string. Now we are ready to write the following function:

```
> labplot <- function (x,y)
 {    xname <- deparse(substitute(x))
```

```
    yname <- deparse(substitute(y))
    plot(x,y, xlab=xname, ylab=yname, main = paste("Plot of",
          yname,"versus", xname))
}
```

a) Study and illustrate the usage of function `labplot()`.

b) From Figure 7.8.1 it also follows that the function `parse()` does the opposite of `deparse()` by converting a character string into an unevaluated expression. The latter unevaluated expression can be evaluated when needed using `eval()`.

7.9 Operators

Execute the following instruction

```
> objects('package:base')[1:31]
```

in order to obtain some examples of operators available in R.

a) ***Operators are special R functions***. Discuss this statement. In what respects do operators differ from ordinary R functions?

b) Write an operator `%E%` to determine the Euclidean distance between two vectors and give an example of its usage. *Hint:* when creating operators with fix or using scripts the name must be given as a character string e.g. fix `("%E%")`.

7.10 Replacement functions

Execute the following instruction

```
> objects('package:base')
```

and notice that some object names appear in pairs with the name of one member of the pair ending in `'<-'`. Examples are `'dim<-'`, `'levels<-'`, `'diag<-'`, `'names<-'`, `'rownames<-'`, `'colnames<-'` and `'dimnames<-'`. Functions having names ending in `'<-'` are called ***replacement*** functions. A replacement function appears on the left-hand side of the assignment symbol using the name without the `'<-'` to replace contents of the objects appearing in its argument list by the contents of the object appearing at the right-hand side of the assignment symbol e.g.:

```
> a <- rownames(X) # Function rownames in action.
> rownames(X) <- 1:nrow(X) # Replacement function 'rownames<-'
                           # in action.
```

How can the object `'diag<-'` be inspected and is it different from the object `'diag'`? Compare the result of the following function calls:

```
> getAnywhere('diag')
> getAnywhere('diag<-')
```

In what respects do replacement functions differ from other functions? In order to write a replacement function the following rules must be met:

i. the function name must end in <-

ii. the function must return the complete object with suitable changes made

iii. the final argument of the function corresponding to the replacement data on the right-hand side of the assignment, must be named *value*

iv. usually a companion function exists having the same name without the '<-'.

As an example, write a replacement function `undefined()` that will replace missing values in a data object with the values on its right-hand side:

```
"undefined<-" <- function (x, codes = numeric(), value)
 { if (length(codes) > 0) x[x %in% codes] <- NA
   x[is.na(x)] <- value
   x
 }
```

The above function can be created or edited using `fix("undefined<-")`. Illustrate the usage of `undefined()`.

7.11 Default values and lazy evaluation

a) The function `match.arg()` is useful for selecting a default value from one of a set of possible values. Consider the following example:

```
> choice <- function(method=c("PCA","CVA","CA","NONLIN"))
  { match.arg(method) }
> choice()
 [1] "PCA"
> choice("CVA")
 [1] "CVA"
> choice("xx")
   Error in match.arg(method) :
   'arg' should be one of "PCA", "CVA", "CA", "NONLIN"
```

b) Functions in the S language are governed by a principle known as ***lazy evaluation*** which means that a default value is not evaluated until it is actually needed within the function body. As a result of lazy evaluation it might happen in a function call that some default values are never evaluated.

7.12 The dynamic loading of external routines

Compiled code can run in some instances much faster than corresponding code in R. The functions `.C()` and `.Fortran()` allow users to make use of programs written in C or Fortran in their R functions. How this is done is illustrated below. Study this example carefully and consult the help files for more details when needed.
First an R function is created to compute the matrix product of two matrices:

```
> matmult <- function (A,B)
 { if(ncol(A) != nrow(B)) stop("A and B not conformable with
                        respect to matrix multiplication \n")
     n <- nrow(A)
     q <- ncol(B)
     Cmat <- matrix(NA, nrow=n, ncol=q)
     for(i in 1:n)
      { for(j in 1:q) Cmat[i,j] <- sum(A[i,] * B[,j])
      }
 Cmat
 }
```

Next a Fortran subroutine is written for performing matrix multiplication. The Fortran code for this subroutine is given below:

```
      SUBROUTINE MATM (A1, A2B1, B2, A, B, OUT)
C     This subroutine performs matrix multiplication.
C     This should be improved with optimized code (such as
C     from Linpack, etc.)
      IMPLICIT NONE
      INTEGER A1, A2B1, B2
      DOUBLE PRECISION A(A1,A2B1), B(A2B1,B2), OUT(A1,B2)
C     DUMMIES
      INTEGER I, J, K
      DO 300,J=1,B2
        DO 200,I=1,A1
          OUT(I,J)=0
          DO 100,K=1,A2B1
            OUT(I,J)=OUT(I,J)+A(I,K)*B(K,J)
100   CONTINUE
200   CONTINUE
300   CONTINUE
      END
```

Next a dynamic link library (.dll) is made from the Fortran subroutine. The easiest way to do this is to use the command `R CMD SHLIB matm.f` from a dos shell. The dll is available as C:\ *matm32.dll* (for 32 bit machines) or C:*matm64.dll* (for 64 bit machines).

Now an R function is to be written where the Fortran code is called:

```
> matmult.Fortran <-function (A,B)
  { if(ncol(A) != nrow(B)) stop("A and B not conformable with
                          respect to matrix multiplication \n")
      n <- nrow(A)
      q <- ncol(B)
      p <- ncol(A)
      Cmat <- matrix(0, nrow=n, ncol=q)
      storage.mode(A) <- "double"
      storage.mode(B) <- "double"
      storage.mode(Cmat) <- "double"
      value <- .Fortran("matm", as.integer(n), as.integer(p),
                        as.integer(q), A, B, matprod=Cmat)
      value$matprod }
```

In order to use `matmult.Fortran()` the correct dll must be loaded into the working directory using the function `dyn.load()`:

```
> dyn.load("full path\\matm64.dll")
            # Change 64 to 32 for 32bit machine
```

Compare the answers and execution time of `matmult()` and `matmult.Fortran()` for different sized matrices.

8 Vectorized programming and mapping functions

In this chapter we continue the study the art of R programming. An important topic is a set of tools operating on objects like matrices, dataframes and lists as wholes.

8.1 Mapping functions to a matrix

a) What is understood by a mapping function and of what use are such functions?

b) The function `apply()`.

i. What three arguments are required?

ii. Suppose the third argument is a function. How are the arguments of this function used within `apply()`?

1) What is the result of the instruction `apply(is.na(x),2,all)`?

2) What is the result of the instruction
`x[ ,!apply(is.na(x), 2,all)]`?

3) What is the result of the instruction

`x[ ,!apply(is.na(x), 2,any)]`?

4) Set the random seed to 137921. Obtain a matrix **A**: 10 × 6 of random n (0, 1) values. Use `apply()` to find the 10% trimmed mean of each row.

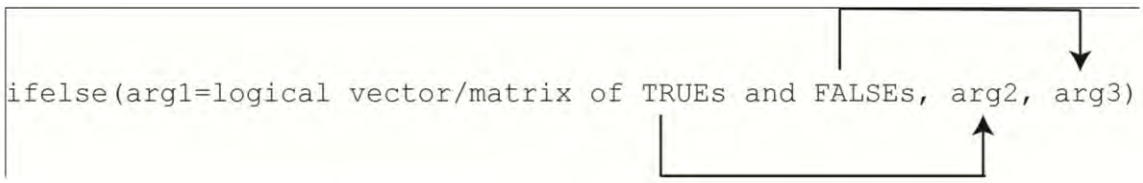
c) The function `sweep()`.

i. What arguments are required?

ii. What are the similarities and differences between the arguments of `sweep()` and `apply()`?

iii. Normalize the columns of a given matrix to have zero means and unit variances using `scale()`, `apply()` and `sweep()`. Which method is the fastest?

d) The function `ifelse()`.
The usage is illustrated in the following diagram.



i. Note the difference between the function `ifelse()` and the control statement: `if – else`.

ii. What arguments are required?

iii. Study the help file in detail.

e) The function `outer()`.
i. What arguments are required?

ii. Revise our previous example of `outer()` when constructing a perspective plot with `persp()`.

f) Work through the following examples and note in particular how the above functions are used together:

i. Find the maximum value(s) in each column of the `LifeCycleSavings` data set.

ii. Use `apply()` together with `cut()` to divide each column of the `LifeCycleSavings` data set into *low*, *medium* and *high*.

iii. Use `apply()` to plot each column of the `LifeCycleSavings` data set against the ratio of `pop75` to `pop15` on the *x*-axis.

iv. Use `apply()` to find the coefficient of variation of each column of the `LifeCycleSavings` data set.

v. Use `apply()` together with `cbind()` and `rbind()` to obtain a table of the minimum and the maximum values of each column of the `LifeCycleSavings` data set.

vi. Repeat (v) using the `airquality` data set with and without the elimination of the `NA`s by using an appropriate function definition in the call to `apply()`.

vii. Use `sweep()` to convert the `LifeCycleSavings` data set into standardized scores. Could `apply()` also be used for this task? Discuss.

viii. Use `ifelse()` to convert negative values in a given vector to zero leaving positive values and missing values unchanged. Illustrate.

8.2 Mapping functions to vectors, dataframes and lists

a) Study the functions `lapply(), sapply() and split()`.

b) Study carefully what is produced by the command

```
> lapply(split(data.frame(state.x77),
  cut(data.frame(state.x77)$Illiteracy,3)),pairs)
```

Note: in order to see all graphs it is necessary to issue the command

```
> par(ask=TRUE)
```

before calling the function lapply().

c) Use `lapply()` to produce histograms of each of the variables in the `state.x77` data set such that each histogram has as title the correct variable name. The *x*- and *y*-axis must also be labelled correctly.

8.3 The functions: `mapply(),rapply()` an `Vectorize()`

Study the helpfiles of `mapply(), rapply()and Vectorize()`.

8.4 The mapping function `tapply()` for grouped data

a) Study the arguments of `tapply()`.

b) Consider the `LifeCycleSavings` data set. Create an object `ddpigrp` that groups the `LifeCycleSavings` data into four groups G1, G2, G3 and G4 such that G1 members have `ddpi` within $(0, 2.0]$, G2 members have `ddpi` within $(2.0, 3.5]$, G3 members have ddpi within $(3.5, 5.0]$, and G4 members have `ddpi` larger than 5.0. Use `tapply()` to obtain the mean aggregate personal savings of each of the groups defined by `ddpigrp`.

c) If it is needed to break down a vector by more than one categorical variable, a list containing the grouping variables is used as the second argument to `tapply()`. Illustrate this by finding the mean aggregate personal savings of the groups in `ddpigrp` broken down by the `pop15` rating.

d) In order to use `tapply()` on more than one variable simultaneously `apply()` can be used to map `tapply()` to each of the variables in turn. Study the following command and its output carefully:

```
> apply(LifeCycleSavings[,c(1,3,4)],2, function(x)
      tapply(x, ddpigrp, mean))
```

e) If `tapply()` is called without a third argument it returns a vector of the same length than its first argument containing an index into the output that normally would be produced . Illustrate this behaviour and discuss its usage.

8.5 The control of execution flow statement `if-else` and the control functions `ifelse()` and `switch()`

a) The primary tool for conditional computations is the *if* statement. It takes the form:

```
if(logical condition evaluating to either TRUE or FALSE)
     {
     First set consisting of one or more R expressions
     }
else
     {
     Second set consisting of one or more R expressions
     }
Expression3
```

b) In the above the `else` and its accompanying expression(s) are optional.

c) `if-else` statements can be nested.

d) Remember that the function `ifelse()` operates on objects as wholes as illustrated below:

```
> xx <- matrix(1:25, ncol=5)
> xx
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

> ifelse(xx < 10, 0, 1)
          [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    1
[2,]    0    0    1    1    1
[3,]    0    0    1    1    1
[4,]    0    0    1    1    1
[5,]    0    1    1    1    1
```

e) Note that the function `match()` can be used as an alternative to multiple `if-else` statements in certain cases. The function `match()` takes as first argument a vector, x, of values to be matched and as second argument, `table`, a vector of possible values to be matched against. A third argument `nomatch = NA` specifies the return value if no match occurs. See example below:

```
> match(c(1:5,3),c(2,3))
 [1] NA 1  2 NA NA  2

> match(c(1:5,3),c(2,3),nomatch = 0)
[1] 0 1 2 0 0 2

> match(c(1:5,3),c(3,2),nomatch = 0)
[1] 0 2 1 0 0 1
```

f) The following example provides an illustration of the usage of `match()`:

```
> month.num <- 5:9
> month.name <- c("May", "June", "July", "Aug", "Sept")
> new.vec <-
     month.name[match(airquality[,"Month"],month.num)]
> out <- data.frame(airquality[ ,1:5],
      MonthName = new.vec, Day = airquality$Day)
```

```
> out[c(1:5,148:153), ]
     Ozone  Solar.R Wind  Temp  Month  MonthName  Day
1      41    190     7.4    67    5      May        1
2      36    118     8.0    72    5      May        2
3      12    149     12.6   74    5      May        3
4      18    313     11.5   62    5      May        4
5      NA    NA      14.3    56    5      May        5
148    14    20      16.6    63    9      Sept     25
149    30    193      6.9    70    9      Sept     26
150    NA    145     13.2    77    9      Sept     27
151    14    191     14.3    75    9      Sept     28
152    18    131      8.0    76    9      Sept     29
153    20    223     11.5    68    9      Sept     30
```

g) The function `switch()` provides an alternative to a set of nested `if – else` statements. It takes as first argument, `EXPR`, an integer value or a character string and as second argument, ..., the list of alternatives. As an illustration:

```
> centre <- function(x, type)
 { switch(type,
          mean = mean(x),
          median = median(x),
          trimmed = mean(x, trim = 0.1))
 }

> x <- rcauchy(10)
> x
    [1] -0.2728273 -0.4464700 -0.0122678 1.7110551 0.7312103
    -0.4521207 0.6121033 1.3171475 -0.9533983 -5.7990911

> centre(x,"mean")
   [1] -0.3564659

> centre(x,"median")
   [1] -0.1425475

> centre(x,"trimmed")
   [1] 0.06542216
```

h) The two logical control operators `&&` and `||` are useful when using `if – else` statements. These two operators operate on logical expressions in contrast to the operators `&` and `|` which operate on vectors/matrices.

8.6 Loops in R

a) `for` loops: The general form is

```
for (name in values)
      { expression(s)
      }
```

This type of loop is useful if it is known in advance how many times the statements in the loop are to be performed. In the above definition values can be either a vector or a list with elements not restricted to be numeric:

```
> for (i in 1:26) cat(i, letters[i],"\n")
> for (letter in letters) cat(letter, \n")
```

Consider a list consisting of several matrices, each with different numbers of rows but the same number of columns. Write an R function that will create a single matrix consisting of all the elements of the given list concatenated by rows.

b) `while loops`: The general form is

```
while (condition)
        { expression(s)
        }
```

This type of loop continues while condition evaluates to TRUE.

c) Control inside loops: `next` and `break`
The command `next` is used to skip over any remaining statements in the loop and continue executing. The command `break` causes the immediate exit from the loop. In nested loops these commands apply to the most recently opened loop.

d) `repeat` loops: The general form is

```
repeat { expression(s)
 }
```

This type of loop continues until a `break` command is encountered.

e) Remember that many operations that might be handled by loops can be more efficiently performed in R by using the subscripting tools discussed earlier.

f) As a further example we will consider the calculation of the Pearson chi-squared statistic for the test of independence in a two-way classification table:

$$\chi_p^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(f_{ij} - e_{ij})^2}{e_{ij}}$$

with $e_{ij} = f_{i.}f_{.j}/f_{..}$ the expected frequencies. This statistic can be calculated in R without using loops as follows:

```
> fi. <- ftable %*% rep(1,ncol(ftable))
> f.j <- rep(1,nrow(ftable)) %*% ftable
> e <- (fi. %*% f.j)/sum(fi.)
> X2p <- sum((ftable-e)^2/e)
```

g) Explicit loops in R can potentially be expensive in terms of time and memory. The functions `apply()`, `tapply()`, `sapply()` and `lapply()` should be used instead if possible. The expected frequencies in the previous example can, for example, be obtained as follows:

```
> e.freq <- outer (apply (ftable, 1, sum), apply (ftable, 2, sum)) / sum(ftable)
```

8.7 The execution time of R tasks

The functions `system.time()` and `proc.time()` provide information regarding the execution of R tasks.

a) `proc.time` determines how much real and CPU time (in seconds) the currently running R process has already take:

```
> proc.time() # called with no arguments
user system elapsed
2.24 1.18 592.60
```

b) `system.time(expr)` calls the function `proc.time()`, evaluates `expr`, and then calls `proc.time()` once more, returning the difference between the two `proc.time()` calls:

```
> system.time(hist(rev(sort(rnorm(1000000)))))
 user system elapsed
 0.39 0.04 0.46
```

Note that user and system times do not necessarily add up to elapsed time exactly.

c) Write the necessary code using `proc.time()` directly to obtain the execution time of `hist(rev(sort(rnorm(1000000))))`.

d) As an application of `system.time()` and `proc.time()` perform the following simulation study: Given a covariance matrix $\mathbf{S}: p \times p$ the task is to compute the corresponding correlation matrix. The execution times of the following three methods are to be compared:

i. Direct elementwise calculation of $r_{ij} = {s_{ij}}\Big/{\sqrt{s_{ii}s_{jj}}}$ using two nested for loops;

ii. Two applications of `sweep()`;

iii. Matrix multiplication where $\boldsymbol{R}: p \times p = diag\left(\boldsymbol{S}^{-\frac{1}{2}}\right)\boldsymbol{S}diag(\boldsymbol{S}^{-\frac{1}{2}})$ and $diag(\boldsymbol{A})$ denotes the diagonal matrix formed from $\boldsymbol{A}: p \times p$ by setting all its off-diagonal elements equal to zero.

Use `var()` and `rnorm()` to compute covariance matrices of different sizes p from samples varying in size n. Study the role of n and p in the effectiveness (economy in execution time) of the above three methods. Display the results graphically. Remember that for valid comparisons the three methods must be executed with identical samples.

8.8 The calling of functions with argument lists

a) The function `do.call()` provides an alternative to the usual method of calling functions by name. It allows specifying the name of the function with its arguments in the form of a list:

```
> mean(c(1:100,500),trim=0.1)
[1] 51
> do.call("mean",list(c(1:100,500), trim=0.1))
[1] 51
```

b) How does `do.call()` differ from the function `call()`?

c) As an illustration of the usage of `do.call()` study the following example:

```
> na.pattern <- function(frame)
  { nas <- is.na(frame)
    storage.mode(nas) <- "integer"
    table(do.call("paste",c(as.data.frame(nas),
                  sep = "")))
  }
```

```
> na.pattern(as.data.frame(airquality))
  000000 010000 100000 110000
   111      5      35      2
```

What can be learned from the above output?

d) What is the difference between `as.integer()`, `storage.mode() <- "integer"`, `storage.mode()` and `mode()`?

8.9 Evaluating R strings as commands

Recall from Figure 7.8.1 that the function `parse(text = "3 + 4")` returns the unevaluated expression `3 + 4`. In order to evaluate the expression use function `eval(): eval(parse(text="3+4"))` returns 7.

8.10 Object-oriented programming in R

Suppose we would like to investigate the body of function `plot()`. We know that this can be done by entering the function's name at the R prompt:

```
> plot
function (x, y, …)
UseMethod("plot")
<bytecode: 0x000000001051f468>
<environment: namespace:graphics>
```

The presence of `UseMethod("plot")` shows that `plot()` is a ***generic function***. The ***class*** of an object determines how it will be treated by a generic function i.e. what ***method*** will be applied to it. Function `setClass()` is used for setting the class attribute of an object. Function `method()` is used to find out (a) what is the repertoire of methods of a generic function and (b) what methods are available for a certain class:

```
> methods(plot)
 [1] plot.acf*              plot.data.frame* plot.decomposed.ts*
 [4] plot.default           plot.dendrogram* plot.density
 [7] plot.ecdf              plot.factor*     plot.formula*
[10] plot.function          plot.hclust*     plot.histogram*
[13] plot.HoltWinters*      plot.isoreg*     plot.lm
[16] plot.medpolish*        plot.mlm         plot.ppr*
[19] plot.prcomp*           plot.princomp*   plot.profile.nls*
[22] plot.spec              plot.stepfun     plot.stl*
[25] plot.table*            plot.ts          plot.tskernel*
[28] plot.TukeyHSD
 Non-visible functions are asterisked
```

```
> methods(class="lm")
  [1] add1.lm*           alias.lm*          anova.lm           case.names.lm*
  [5] confint.lm*        cooks.distance.lm* deviance.lm*       dfbeta.lm*
  [9] dfbetas.lm*        drop1.lm*          dummy.coef.lm*     effects.lm*
 [13] extractAIC.lm*     family.lm*         formula.lm*        hatvalues.lm
 [17] influence.lm*      kappa.lm           labels.lm*         logLik.lm*
 [21] model.frame.lm     model.matrix.lm    nobs.lm*           plot.lm
 [25] predict.lm         print.lm           proj.lm*           qr.lm*
 [29] residuals.lm       rstandard.lm       rstudent.lm        simulate.lm*
 [33] summary.lm         variable.names.lm* vcov.lm*
```

In broad terms there are currently two types of classes in use in R: The old classes or S3 classes and the newer S4 classes. An S4 object can contain one or more ***slots*** which can be accessed using the operator @. Central to the concept of object-oriented programming is that a method can inherit from another method. The function `NextMethod()` provides a mechanism for ***inheritance***.

a) As an example of a generic function study the example in the help file of the function `all.equal()`.

b) R provide many more facilities for writing object-oriented functions. Consult R *Language Definition Manual Chapter 5: Object-Oriented Programming* for further details.

c) A statistical investigation is often concerned with survey or questionnaire data where respondents must select one of several categorical alternatives. The `questdata` below shows the responses made by 10 respondents on four questions. The alternatives for each question were measured on a five point categorical scale. We can refer to the `questdata` dataframe as the full data. This form of representing the data is not an effective way of storing the data when the number of respondents is large. A more compact way of saving the data without any loss in information is to store the data in the form of a ***response pattern*** matrix or dataframe. The first row of `questdata` constitutes one particular response pattern namely (`"b" "c" "a" "d"`). A response pattern matrix (dataframe) shows all the unique response patterns together with the frequency with which each of the different response patterns has occurred. Your challenge is to provide the necessary R functions to convert the full data into a response pattern representation, and conversely to recover the full data from its response pattern representation.

```
>  questdata
   Q1  Q2  Q3  Q4
1  "b" "c" "a" "d"
2  "d" "d" "c" "a"
3  "a" "d" "c" "e"
4  "a" "d" "c" "e"
5  "b" "c" "a" "d"
6  "a" "d" "c" "e"
7  "b" "c" "a" "d"
```

```
8   "d" "d" "c" "a"
9   "c" "b" "a" "e"
10  "b" "c" "a" "d"
```

i. Create the R object `questdata` and then give the following instructions:

```
> unique(questdata[,1])
> duplicated(questdata)
> duplicated(questdata, MARGIN=1)
> duplicated(questdata, MARGIN=2)
> unique(questdata)
> unique(questdata,MARGIN=1)
> unique(questdata,MARGIN=2)
```

ii. Examine Table 3.6.3 and describe carefully the behaviour of the functions `duplicated()` and `unique()`.

iii. Write an R function, say `full2resp` to obtain the response pattern representation of questionnaire data like those given above. Test your function on `questdata`.

iv. Write an R function, say `resp2full` to obtain the full data set given its response pattern representation. Test your function on the response pattern representation of the `questdata`.

8.11 Recursion

Functions in R can call themselves. This process is called ***recursion*** and it is implemented in R programming by the function `Recall()`.

a) As an example we will use recursion to calculate $x(x+1)(x+2)\ldots(x+k)$ with k a positive integer:

```
> recurs.example <- function (x, k)
 { # Function to calculate x(x+1)(x+2)…..(x+k)
  # where k is a positive integer.
  if (k < 0 )
   stop("k not allowed to be negative or non-integer")
  else if( k == 0) x
   else(x+k) * Recall(x,k-1)
 }
```

Investigate if `recurs.example()` works correctly.

b) Explain how recursion works by studying the output of the following function for values of r =1, 2, 3, 4, 5, 6:

```
Recursiontest <- function(r)
      { if(r <= 0) NULL
        else { cat("Write = ", r, "\n")
               Recall(r - 1)
               Recall(r - 2)}
             }
```

c) Use recursion and the function `Recall()` to write an R function to calculate $x!$.

d) Use recursion to write an R function that generates a matrix whose rows contain subsets of size of the first n elements of the vector v. Ignore the possibility of repeated values in v and give this vector the default value of `1:n`.

8.12 Environments in R

Study the following parts from the *R Language definition Manual*: § 3.5 Scope of variables; Chapter 4: *Functions*

Consider an R function `xx(argument)`. Write an R function to add a constant to the correct object (*i.e.* the object in the correct environment) that corresponds to `argument`. In order to answer this question, you must determine in which environment `argument` exists and evaluation must take place in this environment. Possible candidates to consider are the parent frame, the global environment and the search list. Assume that only the first data basis on the search list is not read-only so that in cases where `argument` can be found anywhere in the search list it can be assigned to the first data basis. *Hint*: Study how the following functions work: `assign(), deparse(), invisible(), exists(), substitute(), sys.parent()`.

8.13 "Computing on the language"

Read *R Language Definition Manual Chapter 6: Computing on the language.*

8.14 Writing user friendly functions in R: The function `menu()`

The function `DescribeData()` given below, enables the user to choose, with the help of menus, what statistic must be calculated or what graph must be drawn. It provides a simple example how to write a program for a user that possesses no knowledge about R, but is only interested in using R to successfully complete a task. Note specifically how `menu()` is utilized in this function.

```
DescribeData <- function ()
{ # This function reads in user data.
  # The data must satisfy the following conditions:
  # 1. Available in the text file c:\\Temp\\MyData.txt
  # 2. There musn't be any missing values
  # 3. There are three variables
  # 4. Every sample observation begins on a new line
  # 5. The values of the respective variables must be
  #    separated by a comma
cat("\n", "Descriptive statistics and graph of 3-dim data", "\n", "\n",
"Make sure that the input file satisfies the following conditions:",
"\n",
"1. Single space-separated text file with three columns of data with
no headings",
"\n"," or missing values.", "\n",
"2. Available in the text file c:\Temp\MyData.txt", "\n",
"3. The three variables are continuous", "\n", "\n")
```

```
menu.req <- menu(c(0,1), graphics = F, title="0 = Not correct, 1 =
Correct")
if (menu.req==0) stop("First format the data file correctly")
if (menu.req==1) cat("Proceed and test if the data file exists","\n")
if (!file.exists("c:\\Temp\\MyData.txt"))
      stop("c:\\Temp\\MyData.txt doesn't exist","\n")
else datmat <- as.matrix(read.table("c:\\Temp\\MyData.txt", header=F,
sep=" "))
a <- is.numeric(datmat)
b <- all(!is.na(datmat))
datcorrect <- a & b
if (!datcorrect) stop("c:\\Temp\\MyData.txt exists, but does not
satisfy the requirements")
else menu.req <- menu(c(0,1,2),graphics=F,title="0 = Finished, 1 =
Calculate statistic, 2= Construct graph")
if (menu.req == 0)stop("Task completed")
if (menu.req == 1){
menu.req.1 <- menu(c(1,2),graphics=F,title="1 = Calculate the vector
of means, 2 = Calculate the covariance matrix")
if (menu.req.1 == 1) {cat("Vector of means","\n")
```

```
   return(apply(datmat,2,mean))}
if (menu.req.1 == 2) {cat("Covariance matrix","\n")
   return(var(datmat))}}
if (menu.req == 2)menu.req.2 <- menu(c(1,2,3), graphics=F, title="1 =
Histogram of column 1, 2 = Boxplot of every column, 3 = Scatterplot
matrix")
if (menu.req.2 == 1) hist(datmat[,1])
if (menu.req.2 == 2) boxplot(data.frame(datmat))
if (menu.req.2 == 3) pairs(datmat)
}
```

a) Study carefully how `DescribeData()` works.

b) Write a menu-driven function for reading in an $n \times p$ data matrix. The function must make provision for the user to choose whether Euclidean, Mahalanobis or Manhattan distance must be used to calculate the pairwise distances between all the rows of the data matrix. Illustrate the usage of your function.

8.15 Exercise

Write an R function to determine which positive whole number elements $\leq 10^{10}$ of a given vector are prime and to return these primes. Test this function with randomly generated vectors.

8.16 The function `on.exit()`

What does the function `on.exit()` do?
One use of the special argument ... together with the `on.exit()` function is to allow a user to make temporary changes to graphical parameters of a graphical display within a function. This can be done as follows:

```
function(…)
 { oldpar <- par(…)
   on.exit(par(oldpar))
   or on.exit(par(c(par(oldpar),par(mfrow = c(1,1)))))
   new plot instructions
   ……
 }
```

In the above it is assumed that only arguments of `par()` can be substituted when the function concerned is called. A further use of `on.exit()` is for temporarily changing ***options***.

8.17 Error tracing

Any error that is generated during the execution of a function will record details of the calls that were being executed at the time. These details can be shown by using the function `traceback()`. The function `dump.frames()` gives more detailed information, but it must be used sparingly because it can create very large objects in the workspace. The function `options(error=xx)` can be used to specify the action taken when an error occurs. The recommended option during program development is `options(error = recover)`. This ensures that an error during an interactive session will call `recover()` from the lowest relevant function call, usually the call that produced the error. You can then browse in this or any of the currently active calls to recover arbitrary information about the state of computation at the time of the error. An alternative is to set `options(error = dump.frames)`. This will save all the data in the calls that were active when an error occurred. Calling `debugger()` later on produce a similar result to `recover()`.

The following is a summary of the most common error tracing facilities in R:

`print(), cat()`	The printing of key values within a function is often all that is needed.
`traceback()`	Must be used together with `dump.frames()`.
`options(warn=2)`	Changes *warning* to an error that causes a *dump*.
`options(error=)`	Changes the function that is used for the *dump* action.
`last.dump()`	The object in the .RData that contains a list of calls to *dump*.
`debugger()`	Function to inspect `last.dump` for an error.
`browser()`	Function that can be used within a function to interrupt the latter's execution so that variables within the local frame concerned can be inspected.
`trace()`	Places tracing information before or within functions. Can be used to place calls to the browser at given positions within a function
`untrace()`	Switches all or some of the functions of `trace()` off.

a) Study the *R Language Manual Definition Chapter 9: Debugging* for a summary of error tracing facilities in R . Note especially how the functions `print(), cat(), traceback(), browser(), trace(), untrace(), debug(), undebug()` and `options(warn=2 or error=)` work.

b) Study usage of: `options(error = dump.frames); debugger()`

c) Study usage of: `options(error = dump.frames)`

d) Study usage of the objects `last.dump` and `.Traceback`.

8.18 Error handling: The function `try()`

As an example of the need to be able to handle errors properly consider a simulation study involving a large number of repetitive calculations.

```
Example.8.18.a <- function (iter = 500)
   { select.sample <- function(x)
        { temp <- rnorm(100, m = 50, s = 20)
          if(any(temp < 0))stop("Negative numbers not allowed")
          mean(log(temp))
        }
   out <- lapply(1:iter, function(i) select.sample(i))
   out
  }
```

With `iter` set to a large value, inevitably a call to `Example.8.18.a()` will result in an error message:

```
> Example.8.18.a()
Error in select.sample(i) : Negative numbers not allowed.
```

To see how `try()` can be used make the following change in `Example.8.18.a()`:

```
Example.8.18.b <- function (iter = 500)
   { select.sample <- function(x)
        { temp <- rnorm(100, m = 50, s = 20)
          if(any(temp < 0))stop("Negative numbers not allowed")
          mean(log(temp))
```

```
      }
    out <- lapply(1:iter, function(i)
                          try(select.sample(i), silent = TRUE))
  out
  }
```

A typical chunk of output from a call to `Example.8.18.b()` is

```
> Example.8.18.b(2)
[[1]]
[1] 3.804975
[[2]]
[1] "Error in select.sample(i) : Negative numbers not allowed\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in select.sample(i): Negative numbers not allowed>
```

Notice that execution of `Example.8.18.b` was not halted prematurely. From the above output we can make some final changes to our example function:

```
Example.8.18.c <- function (iter = 500)
  { select.sample <- function(x)
      { temp <- rnorm(100, m = 50, s = 20)
        if(any(temp < 0))stop("Negative numbers not allowed")
        mean(log(temp))
      }
    out <- lapply(1:iter, function(i)
                     try(select.sample(i), silent = TRUE))
    out <- lapply(out, function(x)
                    { if(is.null(attr(x,"condition")))x <- x
                      else x <- attr(x, "condition")
                    })
    Error.report <- lapply(out, function(x) ifelse(!is.numeric(x),
    x, "No Error"))
    Numeric.results     <-     unlist(lapply(out,     function(x)
    ifelse(is.numeric(x),x, NA)))
    list(Error.report = Error.report, Numeric.results = Numeric.
    results)
  }
```

Study the output of a call to `Example.8.18.c` and comment on the merits of `try()` in this example.

9 Reading data files into R, formatting and printing

9.1 Reading Microsoft Excel files into R

The following three ways can be used to read an Excel file into R as an object:

a) The file can be stored as a **.txt* or **.csv* file and then `read.table()`, `scan()` or `read.csv()` can be used to read the file into R.

b) The Excel file can be read directly into R by using the following instructions:

```
> library(RODBC)
> my.object <- odbcConnectExcel("Filename.xls")
      # with the full path included in Filename

> sqlTables(my.object)
      #to obtain information about my.object.

> SENIC.data <- sqlFetch(my.object,"Sheet1")
```

where *Sheet1* is the name of the sheet that contains the data. This name appears under the heading TABLE_NAME when the instruction `sqlTables(my.object)` is executed and must be typed in exactly as it appears, but without the $ sign at the end. The R object `SENIC.data` now contains the required data file.

c) Select the data in Excel. Copy the selected range. In R:
`read.table(file="clipboard")`
Hint: Be careful with empty cells in Excel: some preparation of the Excel file might be needed.

9.2 Reading other data files into R

The R package `foreign()` provides functions for reading data from other packages into R:

```
> library(foreign)
> objects(name="package:foreign")
 [1] "data.restore"   "lookup.xport"   "read.arff"
 [4] "read.dbf"       "read.dta"       "read.epiinfo"
 [7] "read.mtp"       "read.octave"    "read.S"
[10] "read.spss"      "read.ssd"       "read.systat"
[13] "read.xport"     "write.arff"     "write.dbf"
[16] "write.dta"      "write.foreign"
```

Study the helpfiles of these functions for reading into R binary data, SAS XPORT format, Weka Attribute-Relation File Format, the Xbase family of database languages dBase, Clipper and FoxPro, Stata, Epi Info and EpiData files, Minitab portable worksheets, Octave text files, data.dump files that were produced in S version 3[1], SPSS save or export files, SAS data sets to be converted to ssd format[2] and Systat files.

9.3 Sending output to a file

The function `sink("filename")` can be used to divert output that normally appears in the console to a file. The option `options(echo=TRUE)` ensures that the R instructions will also be included in the file. The instruction `sink()` makes output to appear again in the console.
How do the functions `write(x)` and `sink("filename")` differ ? Study the arguments of `write()` thoroughly.

9.4 Writing R objects for transport

The R function `save(file=)` writes an external representation of R objects to the specified file. These objects can be read back from the file using the function `load(file=)`. Study how these two functions work by consulting the help files. Also study how `dump(list, file="outdata")` and `source("outdata")` work in R. The functions `dump()` and `source()` are very useful for transporting R objects between computers.

Note that the instruction `source("instructions")` can be used during an R session to execute the instructions in the external file *instructions*.

9.5 The use of the file .Rhistory and the function `history()`

The file *.Rhistory* is created in the same folder where the *.RData* exists. It can be inspected with any text editor or with MS Word and as such provides an exact record of all activity on the R console (commands window).
Study the help file of the function `history()`.

9.6 Command re-editing

a) Use of the up and down arrows to recall previous commands. Delete, Backspace, Home and End keys for editing.

b) Note the use of the *script* window to execute entire functions or selected instructions only.

9.7 Customized printing

The basic tool for customized printing is the function `cat()`. This function can be used to output messages to the console or to a file. Note the different arguments that are available for `cat()`:

i. By default output is display on the screen; for output to be directed to a file, use argument `file = "file name including path"`.

ii. By default output directed to a file replaces previous contents of the file; use argument `append=TRUE` to append new output to previous contents.

iii. Use `sep="xx"` to automatically insert characters between the unnamed arguments to `cat()` in the output.

iv. To automatically insert newlines in the output use `fill=TRUE`.

v. The `labels=` argument allows insertion of a character string at the beginning of each output line. If `labels` is a vector its values are used cyclically.

- Write today's date as given by the function `date()` in the form

"The date today is: Day of the week, xx, month, 20xx."
as an heading to a file. *Hint*: recall functions `cat()`, `match()`, `substring()`, `paste()`, `replace()`.

9.8 Formatting numbers

a) Study how the functions `round()` and `signif()` together with `cat()` can be used to set the number of decimals that are printed.

b) Study the use of `options(digits=xx)`.

c) Study how the function `format()` works. Note the use of `format()` together with `paste()` and `cat()`.

d) What does `print()` do?

e) Study the help file of `write.table()`.

f) The functions `prmatrix()` or `print()` can be used to output matrices to the console *during execution* of a function. This is very convenient for inspecting intermediate results. Determine how the latter function differs from `cat()`.

g) Note the difference between

```
> colnames(state.x77)
```

and

```
> format(colnames(state.x77))
```

h) Study the following example carefully:

```
> format.mns <- format(apply(state.x77,2,mean))
> format.names <- format(colnames(state.x77))
> descrip.mns <- paste("Mean for variable",
  format.names, " = ", format.mns)
> cat(descrip.mns, fill=max(nchar(descrip.mns)))
Mean for variable % Pop.<15 = 35.0898
Mean for variable % Pop. >75 = 2.2930
Mean for variable Disp. Inc. = 1106.7862
Mean for variable Growth = 3.7576
Mean for variable Savings = 9.6710
```

9.9 Printing tables

Study the example below of how to represent the maximum and minimum value of the variables in the `state.x77` data set in a table with the names of the countries corresponding to the values.

```
> mins <- apply(state.x77, 2, min)
> maxs <- apply(state.x77, 2, max)
> min.name <- character(ncol(state.x77))
> min.name
[1] "" "" "" "" "" "" "" ""

> for(i in 1:8)min.name[i] <-
 rownames(state.x77)[state.x77[,i] == mins[i]][1]
> max.name <- character(8)

> for(i in 1:8)max.name[i] <-
rownames(state.x77)[state.x77 [,i] == maxs[i]][1]

> my.table <- data.frame(mins, min.name, maxs, max.name)
>  dimnames(my.table)  <-  list(names(mins),c("Minimum",  "State  with
Min","Maximum","State with Max"))
> colnames(my.table)[3] <- paste(" ",
                                 colnames(my.table)[3])

> my.table
```

```
            Minimum State with Min       Maximum State with Max
Population  365.00         Alaska       21198.0     California
Income     3098.00    Mississippi        6315.0         Alaska
Illiteracy    0.50           Iowa           2.8      Louisiana
Life Exp     67.96 South Carolina          73.6         Hawaii
Murder        1.40   North Dakota          15.1        Alabama
HS Grad      37.80 South Carolina          67.3           Utah
Frost         0.00         Hawaii         188.0         Nevada
Area       1049.00   Rhode Island      566432.0         Alaska
```

An alternative version of the above table could be obtained with the following instructions:

```
> cat(paste(format(c(" ", "Statistic", " ", names(mins))),
+ format(paste(" ",c(" ", "Minimum"," ",format(mins)))),
+ format(c("State having","Minimum"," ",min.name)),
+ format(paste(" ",c(" ","Maximum"," ",format(maxs)))),
+ format(c("State having","Maximum"," ",max.name))), fill=TRUE)
                 State having State having
Statistic Minimum Minimum Maximum Maximum

   Population     365.00 Alaska                21198.0 California
   Income        3098.00 Mississippi            6315.0 Alaska
   Illiteracy       0.50 Iowa                      2.8 Louisiana
   Life Exp        67.96 South Carolina           73.6 Hawaii
   Murder           1.40 North Dakota             15.1 Alabama
   HS Grad         37.80 South Carolina           67.3 Utah
   Frost            0.00 Hawaii                  188.0 Nevada
   Area          1049.00 Rhode Island         566432.0 Alaska
```

Make the necessary changes in the above lines of code to improve the column spacing

9.10 Communicating with the operating system

Study how the function `system()` works using the DOS instructions: "*time*", "*date*" and "*dir*". *Hint:* First study the help file of the R function `system()` and then the following instructions:

```
> system(paste(Sys.getenv("COMSPEC"),"/c","time /t"),
            show.output.on.console=TRUE,invisible=TRUE)

> system(paste(Sys.getenv("COMSPEC"),"/c","date /t"),
            show.output.on.console=TRUE,invisible=TRUE)
```

```
> system(paste(Sys.getenv("COMSPEC"),"/c","dir c:\\"),
            show.output.on.console=TRUE,invisible=TRUE)
```

The R function `system()` can also be used together with *Notepad* to create a text file during an R session:

```
> system(paste(Sys.getenv("COMSPEC"),"/c","notepad c:\\
temp\\test.txt"),show.output.on.console=TRUE, invisible=TRUE)
```

a) Use `system()` to create a text file without terminating the R session.

b) Use `system()` to write a function `myfile.exists()` that checks if any specified file exists.

9.11 Exercise

1. Construct tables displaying the values of all variables in the *state.x77* data set separately for each region as defined in the R object `state.region`.

2. `Print a table from` the *state.x77* data set such that for each variable, an asterisk is placed after the maximum value for that variable. The numbers must line up correctly.

10 R graphics: Round II

R offers several different types of graphics: *grid* graphics is contained in package `grid`; the package `lattice` contains *trellis* graphics; the package `ggplot2` introduces *ggplot* graphics to be implemented by the function `ggplot()`. In this chapter, further aspects of what is known as *traditional R graphics* are studied.

10.1 Graphics parameters

a) Study the help file of `par()`. Execute `par()` to obtain a list of all the current values of the graphical parameters.

b) How is `par()` used to obtain the current setting of a specific graphics parameter e.g. the parameter `fin`?

```
> par("fin")
```

c) How is `par()` used to change a graphics parameter e.g. `mfrow`?

```
> par(mfrow=c(1,2))
```

d) How do you reset the changed values to their original values? Note the `no.readonly` argument of `par()`. *Hint*: Study the following instructions and there effects carefully:

```
> par('col')
  [1] "black"
```

The current colour for graphics is "black".

```
> temp <- par(col = "blue")
```

Change colour for graphics to "blue".

```
> temp
 $col
 [1] "black"
```

Temp is a list of parameter(s) **BEFORE** change was made.

```
> par('col')
  [1] "blue"
```

Shows that the colour for graphics was indeed changed to "blue".

e) It is sometimes useful to use `par(ask=TRUE)` to instruct R to ask you whether an existing graph should be replaced by a new one.

f) Draw a histogram of variable Ozone in the data set `airquality` where each class interval is randomly represented by a different colour. What happened to the NA values?

10.2 Layout of graphics

a) Review Figure 4.1.1. Note the parameters that are discussed there.

b) Multiple figures on one page: How do the graphical parameters `mfg` and `mfrow` or `mfcol` differ? What are represented in the R data sets `ldeaths`, `mdeaths` and `fdeaths`? Use `mfg` and `mfrow` to obtain Figure 10.2.1. *Hint*: The graphics parameters `mfg` and `mfrow` are used together.

```
> par(mfrow=c(3,2),mfg=c(1,1))
```

The `mfrow` setting reserves three rows and two columns for graphics to be filled row-wise. The `mfg` setting specifies that the next graph will be placed in the position defined by row one column one. Once this graph has been constructed the instruction

```
> par(mfg=c(1,2))
```

will result in the next graph to appear in the position defined by row one and column 2. Next we need the instruction

```
> par(mfrow=c(3,1),mfg=c(2,1))
```

requesting a graph window having three rows and one column with the next graph to appear at position row two (only one column in row two).

c) Note how the meaning of the margins changes when more than one figure is drawn on a page to make provision for an ***outer margin*** surrounding all figures in addition to the ***margin*** surrounding each separate figure.

d) Study how the functions `split.screen(), screen()` and `close.screen()` work as explained in the help facility.

e) Study in detail the usage of the function `layout()` for more complicated arrangements of the graph window. An example of its usage is deferred until later in the chapter.

10.3 Low-level plotting commands

- The functions in Table 10.3.1 are used to edit existing graphs.
- Study these functions carefully.
- Study how the right mouse button is used with R graphs.
- Most plotting tasks require some combination of high-level and low-level plotting commands.

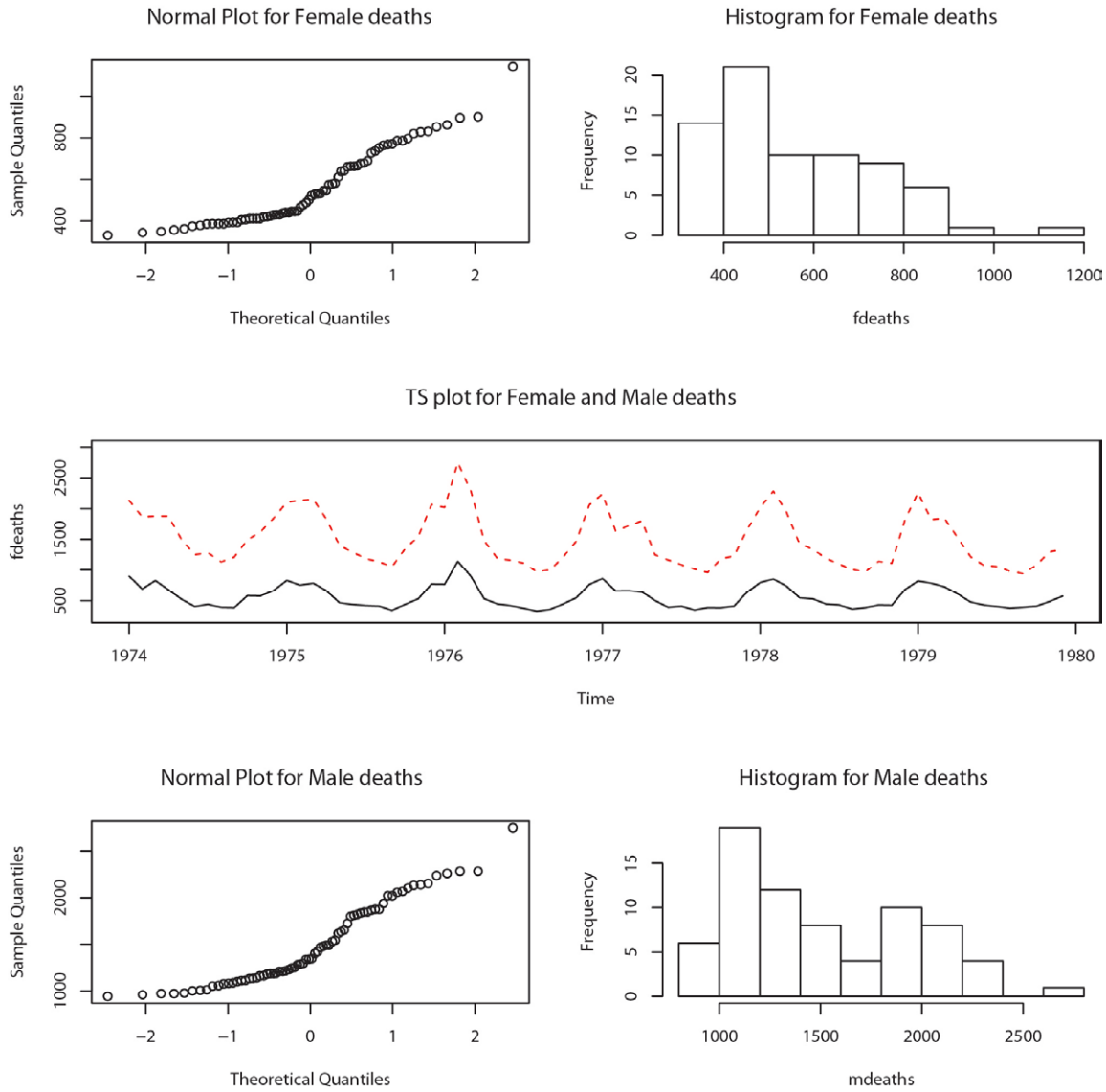


Figure 10.2.1: Plots of the `fdeaths` and `mdeaths` data sets.

Function	Description
abline()	Add regression lines to a plot Also for adding a vertical and horizontal lines to a plot
arrows()	Draw arrow on plot
axis()	Add custom axis to plot
box()	Draw box around plot
legend()	Add a legend to a plot
lines()	Add lines to a plot
mtext()	Write text in margins
points()	Add points to a plot
polygon()	Draw and shade polygons
qqline()	Draw median line on qqnorm plot
rug()	Add data-based marks to an axis
segments()	Draw disconnected line segments
symbols()	Draw symbols on a plot
text()	Add text to a plot
title()	Add titles or axis labels to a plot

Table 10.3.1: Low-level plotting functions.

10.4 Using the plotting commands

10.4.1 Multiple lines or groups of points on the same graph

Study how the function `matplot()` works. Note the functions `matlines()` and `matpoints()`. Study and execute the following example:

```
function ()
  { times <- matrix(0,100,3)
     for(i in 1:100)
      { n <- i * 10000
        s1 <- 1:n
        s2 <- sample(n)
        s3 <- rnorm(n)
        times[i,1] <- system.time(sort(s1))[1]
        times[i,2] <- system.time(sort(s2))[1]
        times[i,3] <- system.time(sort(s3))[1]
      }
     matplot(x=(1:100)*10000,y=times,type="l", lty=1:3,
      col=c("black","green","red"),xlab= "Length",
      ylab="Time in seconds",main="Time for sorting")
 }
```

10.4.2 Multiple lines or groups of points on the same graph but the lines (points) are not all the same length (number)

What technique must be followed? First study the `Cars93` data set in package `MASS`; then study and execute the code below. Experiment with different values of `spar`.

```
function(spar=0.9)
  { require(MASS) # What is the effect of require()?
    oldstate <- par(no.readonly = TRUE) # Describe object
                                          #'oldstate'
    on.exit(par(oldstate)) # Of what use is on.exit()?
    graphics.off()

    cargrp <- Cars93[ , "Type"]
    price <- Cars93[ , "Price"]
    mpg.city <- Cars93[ , "MPG.city"]
    mpg.highway <- Cars93[ , "MPG.highway"]
    plot(price, mpg.city, type = "n", ylim = c(0,
         max(mpg.city)), main = "Fuel Consumption vs Price
         for City Drive", xlab = "Price", ylab = "Miles per
         Gallon in City")
    jj <- 0
    for(i in levels(cargrp))
     { jj <- jj+1
       lines(smooth.spline(price[cargrp==i],
       mpg.city[cargrp==i], spar=spar),lty=jj, col=jj, lwd=2)
     }
    dev.new()
    plot(price, mpg.highway, type = "n", ylim = c(0,
     max(mpg.highway)), main = "Fuel Consumption vs Price
     for Highway Drive", xlab = "Price", ylab = "Miles
     per Gallon on Highway")
    jj <- 0
    for(i in levels(cargrp))
     { jj <- jj+1
       lines(smooth.spline(price[cargrp==i],
         mpg.highway[cargrp==i], spar=spar),lty=jj, col=jj,
         lwd=2)
     }
  }
```

a) Explain the output generated by the above function call.

b) What technique can also be followed in the case of point diagrams?

10.4.3 Adding legends to a graph

a) Study how the function `legend()` and the graphical parameter `usr` work. Study the code used to obtain 10.4.1. Revise the `locator()` function.

b) Use the facts that one USA gallon of liquid is equal to 0.83267 UK (imperial) gallon of liquid and one mile is equal to 1.6093 kilometres to obtain a figure similar to Figure 10.4.1 but with a kilometres per litre scale on the right-hand side that corresponds to the miles per gallon (USA) on highway scale on the left-hand side.

```
function()
   {  require(MASS)
      oldstate <- par(no.readonly = TRUE)
      on.exit(par(oldstate))
      graphics.off()

      cargrp <- Cars93[ , "Type"]
      price <- Cars93[ , "Price"]
```

```
      mpg.city <- Cars93[ , "MPG.city"]
      plot(price, mpg.city, type = "n", ylim = c(0,
        max(mpg.city)), main = "Fuel Consumption vs Price for
        City Drive", xlab = "Price", ylab = "Miles per Gallon
        in
        City")
      char <- substring(as.character(cargrp),1,2)
      text(x=price,y=mpg.city, labels=char, pos=1,cex=0.75)
      labs    <-    paste(substring(levels(cargrp),    1,    2),
levels(cargrp),
        sep=": ")
      legend(x=40,y=45, legend=labs)
    }
```

10.4.4 Multiple plots with identical axes

How can various graphs with identical axes be obtained? Show how this can be done by graphing the sorting time for the three procedures considered in 10.4.1 above in three separate plots in the same graph window.

10.4.5 Providing a single legend for multiple plots

Suppose there were two sorting methods for each of the three situations described in 10.4.1 and 10.4.4 above. How can the three graphs be provided with a single legend without the legend appearing in one of the graphs? Explain in detail.

10.4.6 Changing the plotting character: Common plotting characters in R

Note the use of graphical parameters `pch`. What plotting characters are available? Study the help file of `par() and points()`. Study the plotting characters displayed in Figure 10.4.2 and the code used to produce the figure. How can plotting characters be made to appear in legends?

```
plot(x=rep(1:10,2),y=rep(c(1,2),c(10,10)),pch=0:19,cex=2,
     pty="p",ylim=c(0,3), xlab="",ylab="", xaxt="n",yaxt="n")
```

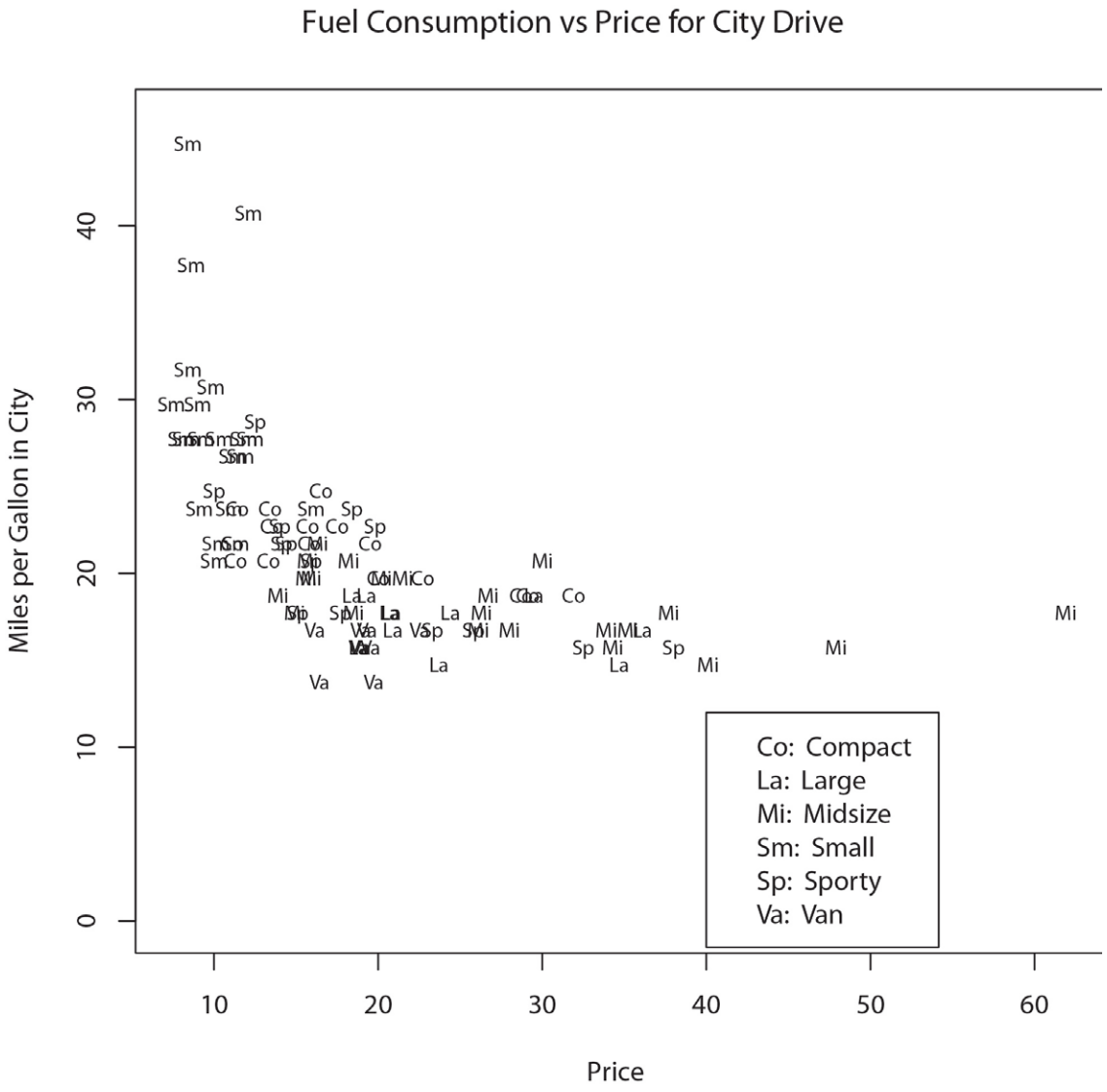


Figure 10.4.1: Illustrating adding a legend to a plot.

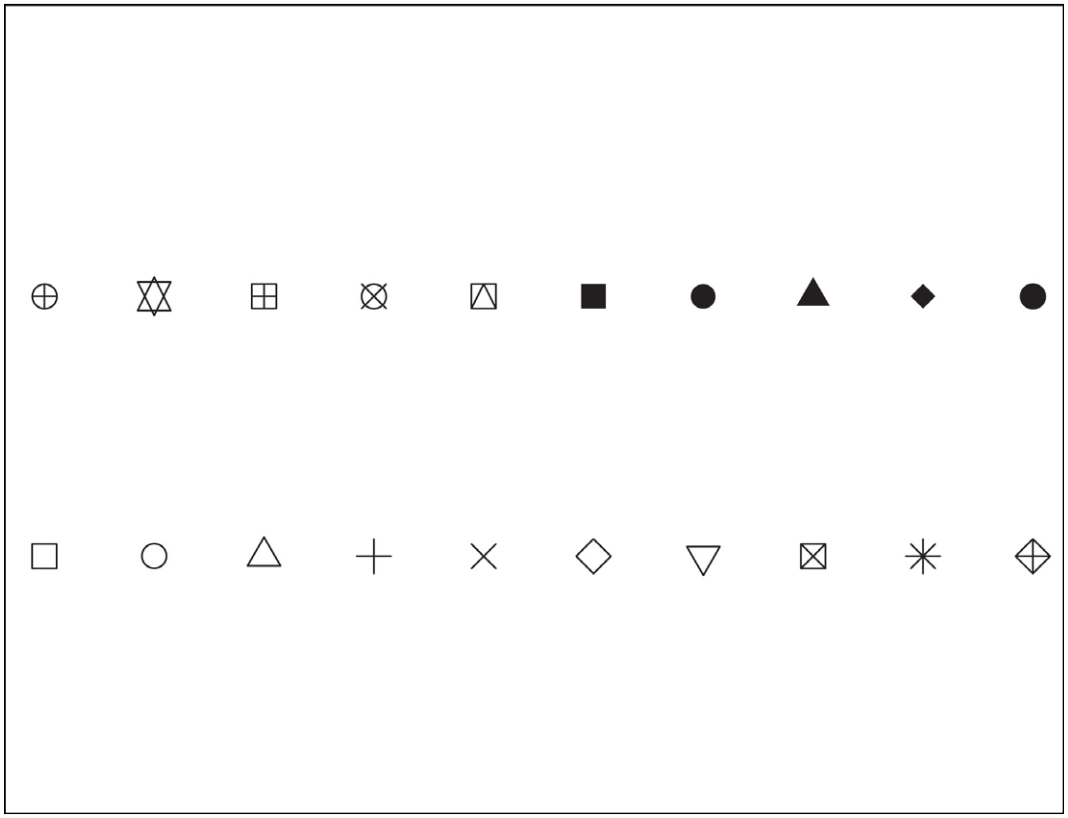


Figure 10.4.2: Some common plotting characters available in R.

10.4.7 Logarithmic axes

The `log()` function and the `log` argument of the `plot()` function are useful in this regard. The `log` argument of the `plot()` function can be specified as `log="x"`; or `log="y"`; or `log="xy"` depending on whether the *x*-axis, the *y*-axis, or both axes should be plotted logarithmically.

10.4.8 Graphs with character strings as the 'scale' on the axis

Figure 10.4.3 illustrates how user defined character strings can appear as calibrations on an axis. Furthermore, this figure illustrates several techniques to fine-tune plots. Study the code resulting in Figure 10.4.3 in detail.

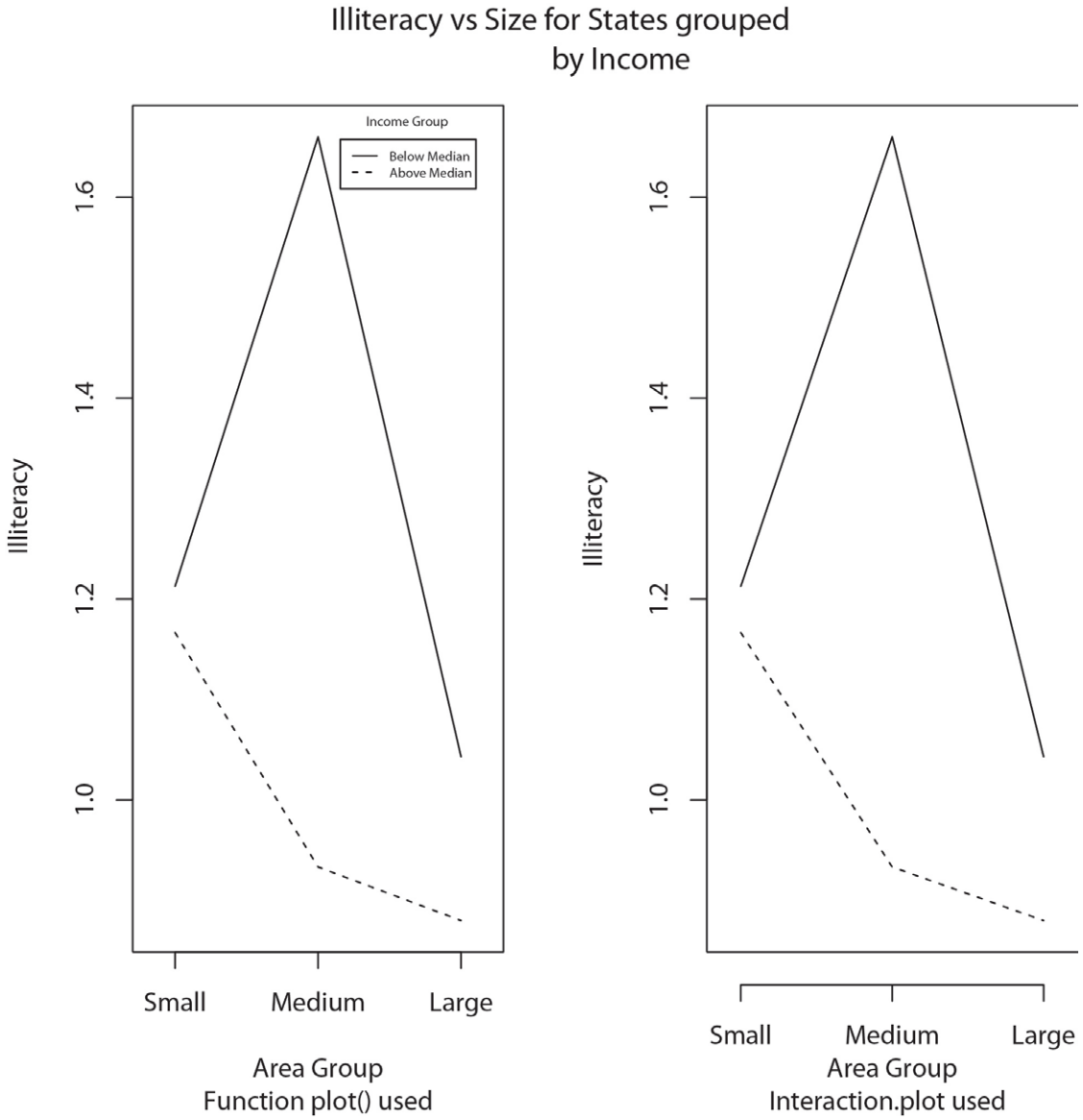


Figure 10.4.3: Figures with character strings as axis calibrations and other enhancements to plots.

```
function()
{ old.state <- par(no.readonly = TRUE)
  on.exit(par(old.state))
  area <- state.x77[, "Area"]
  income <- state.x77[, "Income"]
  area.grp <- cut(area, c(0, quantile(area, c(1/3, 2/3,
           1))),labels = c("Small", "Medium", "Large"))
  income.grp <- cut(income, c(0, quantile(income, c(1/2,
           1))),labels = c("Below Median", "Above Median"))
  mns <- tapply(state.x77[, "Illiteracy"],
                      list(area.grp, income.grp),mean)
  par(mfrow = c(1, 2))
```

```
  plot(c(0.8, 3.2), range(mns), type = "n",
            xaxt="n",xlab = "Area Group", ylab ="Mean
            Illiteracy", sub = "Function plot() used")
  axis(side = 1, at = 1:3, labels = levels(area.grp))
  lines(1:3, mns[, 1])
  lines(1:3, mns[, 2], lty = 2)
  par(usr = c(0, 1, 0, 1))
  legend(0.56,0.96,lty= c(1,2), legend =
  levels(income.grp), cex= 0.5)
  text(0.63, 0.98, adj = 0, "Income Group",cex=.5)
     interaction.plot(area.grp, income.grp, state.x77[,
     "Illiteracy"], xlab = "Area Group", ylab = "Mean
       Illiteracy", sub = "Interaction.plot used",
         lty = 1:2, xtick=T,legend=F)
  par(mfrow=c(1,1))
  par(new=T)
  plot(1:10,1:10,type="n",xlab="",ylab="",axes=F)
  title(main = "Illiteracy vs Size for States grouped
            by Income")
}
```

10.4.9 Customizing bar charts and histograms

a) How can every bar in a bar chart be represented in a different colour and be given separate headings?

b) How can only a line graph without any colours be obtained?

c) How can a probability density function be superimposed on a histogram?

d) How can bar charts be provided with user-defined axes?

Use the `Cars93` data set to answer the above four questions by constructing a figure similar to the one shown in Figure 10.4.4. *Note*: In the Mean MPG plot not all car types are used. If a factor variable is subsetted the original levels will be kept although some of them might not occurred. Hence it might be necessary to create a new factor variable with only the levels that are needed by using `factor()`.

10.4.10 Three-dimensional graphical displays

a) Study how the function `persp()` works.

b) Work through the example code that creates Figure 10.4.5. Apart from the arrow that points to the maximum, different colours must be used to highlight the different aspects of the graph.

c) Provide horizontally and vertically rotated views of the 3D plot.

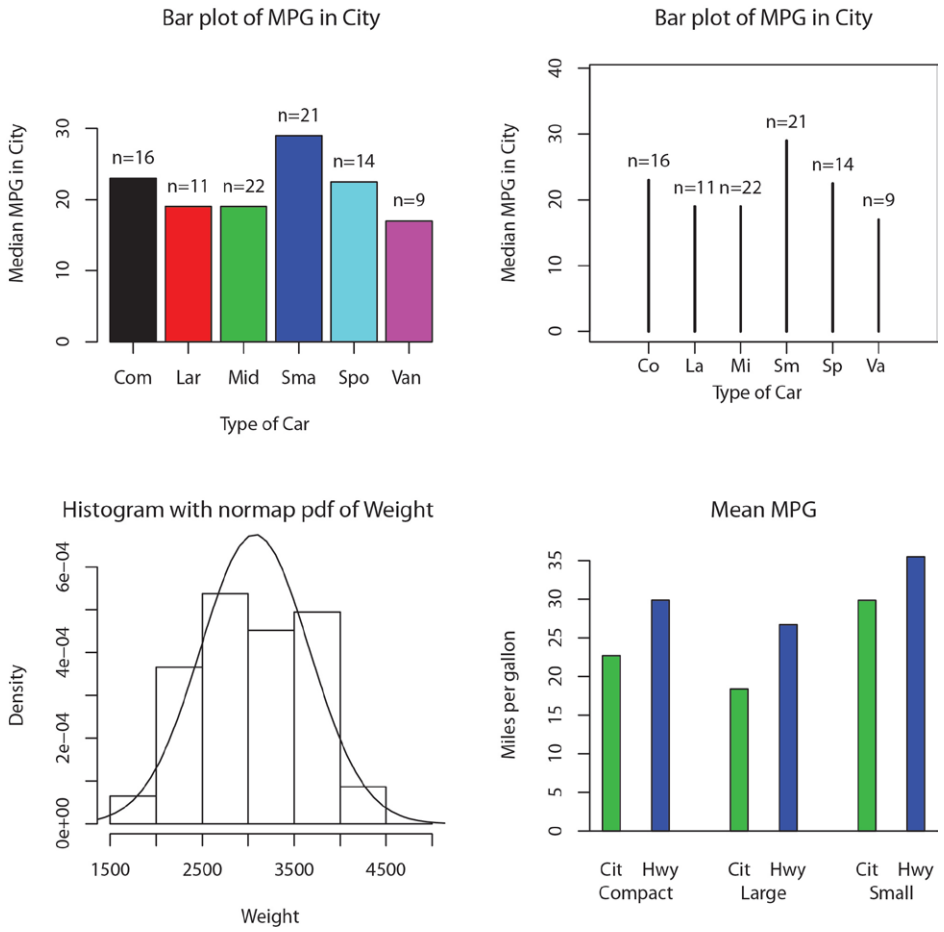


Figure 10.4.4: Enhanced bar charts and histograms.

```
function ()
 { x <- seq(-10, 10, length= 30)
   y <- x
   ff <- function(x,y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
   z <- outer(x, y, ff)
   z[is.na(z)] <- 1
   op <- par(bg = "white")
   persp(x,y,z, theta=30, phi=30, expand=0.5, col="lightblue")
   res <- persp(x,y,z, theta=30, phi=30, expand=0.5, col="lightblue",
   ltheta=120, shade=0.75, ticktype="detailed", xlab="X", ylab="Y",
   zlab = "Z" )
   round(res, 3)
```

```
#--- Add to existing persp plot : ---
#--- Function trans3d() -------------
      trans3d <- function(x,y,z, pmat) {
      tr <- cbind(x,y,z,1) %*% pmat
            list(x = tr[,1]/tr[,4], y= tr[,2]/tr[,4])
       }
# ------------------------------------
   z1 <- ff(1e-10,1e-10)
   transfrm <- trans3d(c(0,-2.5),c(0,5),c(z1,z1),res)
   arrows(transfrm$x[1], transfrm$y[1], transfrm$x[2], transfrm$y[2],
   length=0.1, code=1)
   text(transfrm$x[2], transfrm$y[2]+0.02, "Maximum occurs here")
   return(z1)
}
```

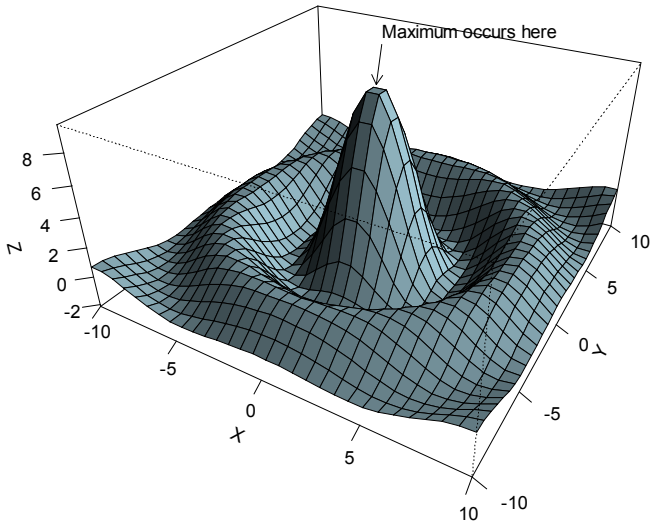


Figure 10.4.5: Annotated 3D perspective plot.

10.4.11 Diagrams

Use R to draw a simple flow diagram. The diagram must contain at least one rectangle, one square, one circle and one triangle. Furthermore, there must be straight and curved lines as well as text describing the different elements. *Hint:* Study how the functions `arrows()`, `lines()`, `text` and `symbols()` work as discussed in their respective help facilities.

10.4.12 Annotating graphics with special symbols

Construct a graph of a normal(0, 1) density function. Give as a title to the plot the expression *Density of a normal random variable with* $\mu = 0$ *and* $\sigma^2 = 1$. *Hint*: Consult the help file of `plotmath`. Within the plot draw an arrow to the density and label it $\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$.

10.5 Exact distances in graphics

a) Obtain a random sample of size 50 from a bivariate normal distribution with $n(50, 20)$ marginals and a correlation coefficient of 0.90.

- Present the data in the form of a scatterplot.
- Next, write an R function to perform the following task on the scatterplot:
  - Choose an arbitrary point and label it "A".
  - Draw a line connecting A to a circle with centre exactly 25mm away from A. The diameter of the circle must be exactly 40mm.
  - Label the centre point of the circle with a "B".

- Use a ruler to check the length of the connecting line and diameter of the circle.

- Obtain a print copy of the graph and check the lengths again.

Hint: Study the help file of function `par()`.

b) Use R to make a ruler calibrated in centimetres from zero to 15 cms.

10.6 Multiple graphics windows in R

a) Study how the following instructions work to control multiple graphics windows in R :

```
dev.list()    dev.set()     dev.next()
dev.cur()     dev.copy()    dev.prev()
dev.off()     dev.ask()     graphics.off()
```

b) Study how the function `windows()` work in R.

c) Study the information that R gives via the execution of `help.search("graph")`.

10.7 More complex layouts

Study the graphical requirements needed for constructing Figure 10.7.1 and how to code these requirements.

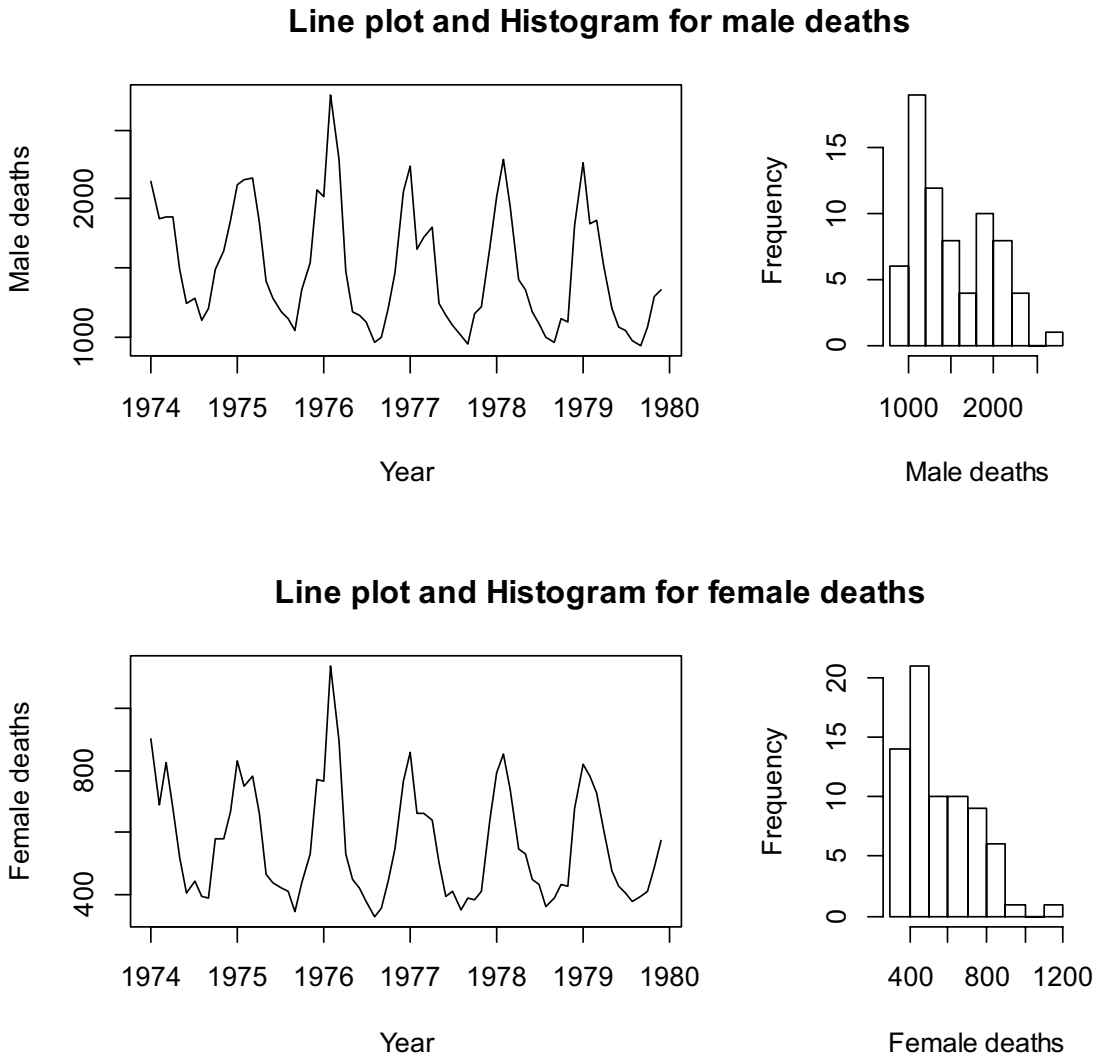


Figure 10.7.1: A complex graphics layout.

```
function ()
 { old.state <- par(no.readonly = TRUE)
   on.exit(par(old.state))
   par(omd=c(0,0.66,0,1), mfcol=c(2,1))
   ts.plot(mdeaths,xlab="Year",ylab="Male deaths")
   ts.plot(fdeaths,xlab="Year",ylab="Female deaths")
   par(omd=c(0.66,1,0,1),mfcol=c(2,1), mfg=c(1,1), new=TRUE)
   hist(mdeaths, xlab="Male deaths", ylab="Frequency",main="")
   hist(fdeaths, xlab="Female deaths",ylab="Frequency",main="")
   par(omd=c(0,1,0,1), mfcol=c(1,1))
   title("Line plot and Histogram for male deaths")
   par(omd=c(0,1,0,0.5), mfcol=c(1,1))
   title("Line plot and Histogram for female deaths") }
```

10.8 Dynamic 3D graphics in R

- Study the R package `rgl`.
- Attach library *rgl* to the search path and then issue the R command example(plot3d). Use the mouse buttons to rotate and zoom the *rgl* graph.
- Next, issue the R command example(surface3d) and interactively explore the 3D figure.

10.9 Animation

Study the following two functions in detail:

```
anim1 <- function (sleep=0.05)
 { # Press ESC to end animation
   n <- 40
   t <- seq(0,2*pi,length=n)
   x <- cos(t)
   y <-sin(t)
   for (i in 1:n)
    { plot.new()
      plot.window(c(-1,1), c(-1,1),asp=1)
      points(x[i],y[i],pch=16,cex=2)
      Sys.sleep(sleep)
      #Sys.sleep() suspends execution of R expressions
      #for a given number of seconds
    }
   Recall(sleep)
 }

anim2 <- function (sleep=0.01)
 { for (i in seq(from=1,to=3, by =0.01))
     { plot.new()
       plot.window(c(1,16),c(1,16),asp=1)
       arrows(2*i,2*i,4*i,4*i)
       Sys.sleep(sleep)
     }
   Recall(sleep)
 }
```

Write an R function to show a wheel with two spokes moving forward with adjustable speed.

10.10 Exercise

1. In many real life situations it is necessary to identify an object when only limited information is available. The following shows how such a problem can be empirically investigated.
   The function `persp()` used for constructing Figure 10.4.5 requires a regular pattern of *x* and *y* coordinates. If such a pattern is not available it is necessary to interpolate e.g. with `interp()` (available in package *akima*) using the available values.
   Use the function `expand.grid()` to create a grid of regularly spaced *x* and *y* values and evaluate the "sombrero" function of Figure 10.4.5 at each of these points.
   Now use `sample()` to randomly sample points from that grid and then the `interp()` function to interpolate the values of *z* throughout the grid.
   Finally, use `persp()` to construct a plot of the interpolated values.
   What fraction of data is needed in the sample to get a good representation of the true shape of the data. *Hint*: since `persp()` does not accept `NA`s replace `NA`s with the minimum of the non-missing *z* values.

2. Use `locator()` and write a function to allow placing a legend with a pointing device anywhere on an existing plot.

3. Use the `state.x77` data set to construct a scatterplot of `Illiteracy` as a function of `Income`. Now construct a second scatterplot of the same data but with the origin on the right-hand side of the *x*-axis. In order to complete this task it is necessary that the values on the *x*-axis decrease from the left-hand side to the right-hand side.

11 Statistical modelling with R

This chapter is an introduction to statistical modelling with R. There are a substantial number of dedicated R modelling packages available. In this chapter only the basics of statistical modelling with R are considered. However, a thorough understanding of these principles is an invaluable aid in mastering any of the available R packages on statistical modelling and related topics.

11.1 Introduction

Some of the aims of statistical modelling are: assessing the relative importance of a set of variables in relation to another variable(s); predicting the value of a dependent variable from those of a set of predictor variables; defining a statistical relationship between the dependent and the independent variables.

Here the following generic functions are considered.

Function	Model	Package
`lm()`	Linear models	`stats`
`aov()`	Analysis of variance	`stats`
`glm()`	Generalized linear models	`stats`
`gam()`	Generalized additive models	`gam mgcv`
`loess()`	Local regression models	`stats`
`rpart()`	Tree-based models	`rpart`
`nls()`	Nonlinear models	`stats`

Table 11.1.1: Generic modelling functions in R.

11.2 Data for statistical models

a) Note the difference between ***factors*** (categorical or classification variables), ***regressors*** (continuous or ***numeric*** variables) and frequency data.

b) Why must the data be in the form of a dataframe when statistical modelling is carried out with R?

c) How are the functions `factor()`, `is.factor()` and `is.ordered()` used to set or determine the class attributes of variables that must behave as factors in further analyses? Note especially the usage of argument `'ordered = '` of function `factor()`.

11.3 Expressing a statistical model in R

a) What role does the tilde-operator (~) play in statistical models in R?

b) Study how the operators in Table 11.1 work. These operators have a different meaning when they appear in a statistical model.

Operator	Algebraic meaning	Meaning in formula after ~
+	Addition	Add a term
−	Subtraction	Remove or exclude a term
*	Multiplication	Main effects and interactions
/	Division	Main effect and nesting
:	Sequence	Interaction
^	Exponentiation	Limit depth of interactions
`%in%`	Matching	Nesting

Table 11.3.1: Operators used in model formulae after the ~.

c) In order to ensure that the operators in Table 11.3.1 have their usual meaning when appearing at the right-hand side of ~ in a statistical model use the `I()` function e.g. `I(a+b)`?

d) Note how ***regression terms***, ***main effects***, ***interaction effects***, ***higher-order interaction effects***, ***nested effects*** and ***covariate terms*** are specified.

e) The following summary illustrates the correspondence between the ***model notation*** of R and the ***algebraic notation*** of models for factors *a* and *b* with numeric (continuous) variable *x*:

Model notation	Algebraic notation
y~a+x	$y = \mu + \alpha_a + \beta x + \varepsilon$
y~a+b+a:b	$y = \mu + \alpha_a + \beta_b + \gamma_{ab} + \varepsilon$
y~a+a:x	$y = \mu + \alpha_a + \beta_a x + \varepsilon$
y~a*x	$y = \mu + \alpha_a + \beta x + \gamma_a x + \varepsilon$
y~-1+a/x	$y = \alpha_a + \beta_a x + \varepsilon$

f) Every numeric variable on the right-hand side of ~ generates one coefficient to be estimated in a fitted model; each ***level*** of a factor variable generates one coefficient to be estimated in a fitted model. What is an ***estimable function***?

g) Note how transformation of variables can be achieved and how model formulae are stored as R objects of class ***formula***.

11.4 Common arguments to R modelling functions

a) The ***formula*** argument. Study how this argument and the period operator work. If a dataframe is specified as an argument to the modelling function a period on the right-hand side of the ~ indicates that the additive effects of all the variables in the dataframe (except those used at the left-hand side of the ~) are included in the model specification.

b) The ***data*** argument. Study how this argument works. Note that this argument will accept any R expression that evaluates to a dataframe. *Important*: if a dataframe is specified in the data argument (i) variables may be referred to by their names; (ii) the dataframe is searched first for a name before the global environment and the rest of the search path; (iii) any other existing object can also be used as a variable.

c) The ***subset*** argument. Note how statistical models can be fitted to subsets of data.

d) The ***weights*** argument. Study how this argument works especially when iterative procedures are used.

e) The ***na.action*** argument. This argument controls how missing values are handled.

f) The ***control*** argument. What is the purpose of this argument?

11.5 Using the statistical modelling objects

Note: each of the modelling functions returns an object whose class attribute has been set to the name of the function e.g. the value of `lm()` is set to be of class `lm`. A set of generic functions are available to provide access to the information contained in them.
To see what is available for a particular modelling function use methods(class=function) e.g.

```
> methods(class=lm)
 [1] add1.lm*          alias.lm*             anova.lm*
 [4] case.names.lm*    confint.lm            cooks.distance.lm*
 [7] deviance.lm*      dfbeta.lm*            dfbetas.lm*
[10] drop1.lm*         dummy.coef.lm         effects.lm*
[13] extractAIC.lm*    family.lm*            formula.lm*
[16] hatvalues.lm*     influence.lm*         kappa.lm
[19] labels.lm*        logLik.lm*            model.frame.lm*
[22] model.matrix.lm   nobs.lm*              plot.lm*
[25] predict.lm        print.lm*             proj.lm*
[28] qqnorm.lm*        qr.lm*                residuals.lm
[31] rstandard.lm*     rstudent.lm*          simulate.lm*
[34] summary.lm        variable.names.lm*    vcov.lm*
```

Function	Description
`add1()`	Add all possible single terms in a model
`anova()`	Return a sequential analysis of variance table or a comparison of two hierarchical models
`coef()` or `coefficients()`	Extract estimated coefficients
`deviance()`	Return the deviance
`df.residual()`	Return the residual degrees of freedom
`drop1()`	Drop all possible single terms in a model
`fitted()` or `fitted.values()`	Extract fitted values
`formula()`	Extract the formula on which the model is based
`influence()`	Extract several types of influence measures
`plot()`	Construct relevant plots.
`predict()`	Calculate predictions (means) optionally with standard errors
`print()`	Extract information about the analysis
`resid()` or `residuals()`	Extract residuals
`step()`	Stepwise selection of a model using an AIC criterion
`summary()`	Extract information about the analysis
`update()`	Modify and/or refit models

Table 11.5.1: Functions for Model Objects.

Note: Do not confuse function `anova()` which operates on an lm object with function `aov()` for fitting an analysis of variance model.

Recall: Asterisked functions are non-visible. Such functions can be accessed using for example

```
> getAnywhere(add1.lm) ↲
```

a) Note the use of `print()`, `summary()` and `plot()` to retrieve information about model objects.

b) Study the functions that are used to retrieve information about model objects or to modify them as summed up in Table 11.5.1.

11.6 Usage of the function `with()`

Study the help file of function `with()` and take note specifically of its usage when calling `lm()`.

11.7 Linear regression and anova

a) Consider the `Cars93` data set available in package `MASS`. Use the `lm()` function to perform a regression analysis of `City fuel consumption` as a function of a constant term, `Length`, $(\texttt{Rev.per.mile})^{-1}$, `Weight`, `RPM` and $(\texttt{Horsepower})^{-1}$. Note how the function `update()` works. Illustrate the use of the functions `drop1()` and `add1()`:

```
> lm.city <- lm(MPG.city ~ 1+ Length + I(1/Rev.per.mile)+ Weight
+ RPM + I(1/Horsepower), data=Cars93)
> summary(lm.city)
Call:
lm(formula = MPG.city ~ 1 + Length + I(1/Rev.per.mile) + Weight
+ RPM + I(1/Horsepower), data = Cars93)
Residuals:
                   Estimate Std.Error t value Pr(>|t|)
(Intercept)      -2.913e+00 1.076e+01  -0.271 0.787288
Length            4.115e-02 3.393e-02   1.213 0.228518
I(1/Rev.per.mile) 7.439e+03 4.760e+03   1.563 0.121750
Weight           -3.345e-03 1.221e-03  -2.740 0.007448 **
RPM               2.750e-03 7.812e-04   3.521 0.000688 ***
I(1/Horsepower)   1.287e+03 2.379e+02   5.410 5.48e-07 ***
---
Signif.codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.677 on 87 degrees of freedom
Multiple R-squared: 0.7855, Adjusted R-squared: 0.7732
F-statistic: 63.72 on 5 and 87 DF, p-value: < 2.2e-16

> lm.city.front <- update(lm.city, subset = DriveTrain=="Front")
# update() is used here to restrict the model fitted in
# lm.city to cars with front wheel drives.
> summary(lm.city.front)
Call:
lm(formula = MPG.city ~ 1 + Length + I(1/Rev.per.mile) + Weight
+ RPM + I(1/Horsepower), data = Cars93, subset = DriveTrain
== "Front")
Residuals:
    Min     1Q  Median    3Q     Max
-5.6598 -1.4476 0.0102 0.9468 13.7023
Coefficients:
                   Estimate Std.Error t value Pr(>|t|)
(Intercept)       1.599e+00 1.418e+01   0.113 0.910571
Length            2.711e-02 4.875e-02   0.556 0.580093
I(1/Rev.per.mile) 7.895e+03 6.518e+03   1.211 0.230462
Weight           -3.808e-03 1.707e-03  -2.230 0.029406 *
RPM               2.672e-03 1.016e-03   2.630 0.010784 *
I(1/Horsepower)   1.246e+03 3.070e+02   4.058 0.000143 ***
---
```

```
Signif.codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.93 on 61 degrees of freedom
Multiple R-squared: 0.7656, Adjusted R-squared: 0.7464
F-statistic: 39.85 on 5 and 61 DF, p-value: < 2.2e-16

> drop1(lm.city, . ~ Length)
# drop1(lm object, scope) shows the effect of dropping the
# Length term from the fitted model
Model:
MPG.city ~ 1 + Length + I(1/Rev.per.mile) + Weight + RPM +
I(1/Horsepower)
       Df Sum of Sq RSS      AIC
<none>               623.27 188.92
Length 1     10.536 633.80 188.48
> drop1(lm.city)
# Shows the effect of dropping each term in turn from the
# fitted model
Model:
MPG.city ~ 1 + Length + I(1/Rev.per.mile) + Weight + RPM +
I(1/Horsepower)
```

```
                   Df Sum of Sq    RSS    AIC
 <none>                         623.27 188.92
 Length             1    10.536 633.80 188.48
 I(1/Rev.per.mile)  1    17.495 640.76 189.50
 Weight             1    53.796 677.06 194.62
 RPM                1    88.801 712.07 199.31
 I(1/Horsepower)    1   209.659 832.93 213.89
> lm.city.const <- lm(MPG.city ~ 1, data=Cars93)
> anova(lm.city.const)
Analysis of Variance Table
Response: MPG.city
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 92 2905.6 31.582

> add1(lm.city.const, . ~ Length+I(1/Rev.per.mile) +
    Weight + RPM + I(1/Horsepower))
# add1(lm object, scope) shows the effect of adding the
# terms in scope individually in turn to the model in lm
# object
Model:
MPG.city ~ 1
                   Df Sum of Sq     RSS    AIC
 <none>                        2905.57 322.09
 Length             1   1289.71 1615.86 269.52
 I(1/Rev.per.mile)  1   1090.68 1814.89 280.32
 Weight             1   2065.52  840.05 208.68
 RPM                1    382.96 2522.61 310.94
 I(1/Horsepower)    1   1927.30  978.27 222.85
```

b) Is the object `a <- as.data.frame(matA)` identical with the object `a1 <- data.frame(matA)`?

c) Once again, consider the `Cars93` data set: Consider cars manufactured by Buick, Chevrolet, Oldsmobile and Pontiac. Use the median of `Weight` to create two car weight groups.

   i. Construct an interaction plot to study the interaction between `Manufacturer` and `Weight` with respect to the highway fuel consumption. Interpret the graph.

   ii. Use the function `aov()` to perform a two-way anova with highway fuel consumption as dependent variable and as independent (explanatory) variables weight-group and manufacturer. What do you conclude about the main effects and the interaction effects?

   iii. Repeat (ii) using `lm()`.

d) Study the help file of the `whiteside` data set available in package `MASS`. Plot the gas consumption as a function of the temperature. Use different plotting characters and colours for the two levels of the factor variable `Insul`. Add to the graph the regression lines of gas consumption on the temperature for the two levels of `Insul`. What do you conclude? Now test for the parallelism of the two regression lines using `lm()` with terms for `Insul`, `Temp` and `Insul:Temp`. Discuss the results of the analysis.

11.8 The function `glm()`

Consider the following data set

```
TreatA TreatB Counts
    1      1     18
    1      2     17
    1      3     15
    2      1     20
    2      2     10
    2      3     20
    3      1     25
    3      2     13
    3      3     12
```

Why is a two-way anova not appropriate here? Use `glm()` with `family=poisson()` to fit the following generalized linear models to the data:

`Counts ~ TreatA + TreatB` and `Counts ~ TreatA * TreatB`

What are your conclusions?

11.9 The function `gam()`

a) Describe briefly what is understood by a ***Generalized additive model***.

b) Inspect the help file of the `stackloss` data set .

c) Execute the following R code:

```
> stack.data <- data.frame(stackloss)
> names(stack.data) <- c("AirFlow","WaterTemp", "AcidConc","Loss")
> stack.gam <- gam::gam(Loss~s(AirFlow) + s(WaterTemp) +
  s(AcidConc), control=gam.control(bf.maxit=50), data=stack.
  data)
> summary(stack.gam)
> par(mfrow=c(3,1))
> plot(stack.gam)
```

Comment on the results.

d) Inspect the help file of the `ethanol` data set available in package `lattice`.

i. Plot `NOx` versus `E`.

ii. Use `lm()` and fit a straight line to the data. Add this line to the plot in (i).

iii. Use `lm()` and fit a second degree polynomial to the data. Add this line also to the plot in (i).

iv. The function `lm()` is used for finding a ***global fit*** to a data set. Describe briefly the use of B-splines for finding a ***local fit*** to a data set.

v. Use `gam()` in package `mgcv` and fit a B-spline to the data. The following code can be used for that:
`mgcv::gam(NOx ~ bs(E, df=xx), data=ethanol).`
Experiment with various values for `xx`.

i. Finally, add a legend to the plot.
Comment on the above results.

11.10 The function `loess()`

a) Describe briefly what is understood by a ***Local regression model*** using the help file of the function `loess()`.

b) Execute the following R code:

```
> stack.loess <- loess(Loss ~ AirFlow * WaterTemp, data=stack.
data)
> summary(stack.loess)
> names(stack.loess)
 [1] "n"         "fitted"          "residuals" "enp"
 [5] "s"         "one.delta"       "two.delta" "trace.hat"
 [9] "divisor"   "pars"            "kd"        "call"
[13] "terms"     "xnames"          "x"         "y"
[17] "weights"
> stack.loess$x
> coplot(stack.loess$y~stack.loess$x[,"AirFlow"]|
           stack.loess$x[,"WaterTemp"],panel=points)
> dev.new()
> coplot(stack.loess$y~stack.loess$x[,"WaterTemp"]|
           stack.loess$x[,"AirFlow"],panel=points)
```

Comment on the results.

11.11 The function `rpart()`

a) Describe briefly what is understood by ***Regression and Classification Trees*** using the help file of the function `rpart()` in package `rpart`.

b) Study the help file of the `kyphosis` data set available in package `MASS`.

c) Execute the following R code:

```
> fit <- rpart(Kyphosis ~ Age + Number + Start,
               data=kyphosis)
> plot(fit)
> text(fit, use.n=TRUE,xpd=NA)
```

Comment on the results.

11.12 Nonlinear regression and the function `nls()`

The asymptotic regression model has the form:

$$y = \alpha + \beta e^{\gamma x}$$

When $\alpha > 0, \beta < 0$ and $\gamma < 0$ the above model becomes Mistcherlich's model of the *law of diminishing returns*. According to the law of diminishing returns the yield, *y*, initially increases quickly with increasing values of *x*, but then the yield slows down and finally levels off just below the value of *α*. The parameters $\alpha > 0, \beta < 0$ and $\gamma < 0$ have the following meaning:
α represents the upper asymptote for yield
β is the difference between the value of *y* when *x* = 0 and the upper asymptote
The asymptotic regression model may be reparameterized such that β is replaced by δ – *α* where δrepresents the yield when *x* = 0. A special case of the asymptotic regression model occurs when *δ* = 0:

$$y = \alpha - \alpha e^{\gamma x} = \alpha(1 - e^{\gamma x})$$

The asymptotic regression model represents a nonlinear regression model and in general no closed form solution exists for estimating its parameters. The R function `nls()` can be used to estimate the parameters of the model. When function `nls()` is employed starting values for the iterative estimating procedure are necessary.

Choosing starting values

1. A starting value for *α* can be found by looking at a scatterplot of *y* as a function of *x* and then chooses as starting value just larger than the largest value of *y*.
2. β is the difference between the value of *y* when *x* = 0 and the upper asymptote. A reasonable starting value is the minimum value of *y* minus a.
3. γ can be roughly initially estimated by the negative of the slope between two "well separated" points on the plot.
   a) Study the help file of `nls()`. Notice carefully how `nls()` is called.
   b) The data below is the *Yield* of a product after a chemical treatment of different time duration. Three replicates were made at each treatment time.

Time (Hrs)	Rep1	Rep2	Rep3
0	0	0	0
3	0.0846	0.0556	0.0501
6	0.1072	0.0604	0.0545
12	0.1255	0.059	0.0705
18	0.1671	0.0799	0.0687
24	0.1988	0.0958	0.0655
48	0.2927	0.1739	0.1075
72	0.3713	0.1910	0.1418
96	0.4773	0.2669	0.1725
168	0.6158	0.3584	0.3224
336	0.7297	0.4339	0.3322
504	0.8083	0.4816	0.3309
672	0.7019	0.4497	0.3798
840	0.6038	0.4851	0.3757
1008	0.7386	0.5332	0.3798

Plot the data.

c) The above plot suggests a relationship of the form:

$$y = \alpha - \alpha e^{\gamma x} = \alpha(1 - e^{\gamma x})$$

Explain the term: ***nonlinear regression***. Use `nls()` to estimate the parameters α and γ in the above model. What effect does the changing of the start values have on the result of `nls()`? Experiment with different start values for the parameters. Add the estimated line to the plot in (b) and interpret the graph.

11.13 Normal quantile plot

Consider the the `lm` objects `lm.city` and `lm.city.front` created in Section 11.7. Obtain normal quantile plots of the residuals associated with these two objects and use `qqline()` for adding straight lines to the respective plots. Interpret your plots.

11.14 A coplot with two conditioning variables

a) Consider the `state.x77` data set. Obtain a plot of `Illiteracy` and `Life expectancy` conditional on `Income` and `Area`. Interpret.

b) When producing a coplot it is helpful to draw lines or curves in each of the dependency panels, to make the relationship easier to appreciate. Use the *panel* argument of `coplot()` to draw in each panel

i. a straight line representing the linear regression between the variables;

ii. a curve representing the polynomial regression;

iii. a spline using `smooth.spline()`.

Hint: Graphical arguments are passed to `coplot()` by writing a suitable panel function as 'value' for argument `panel` of `coplot()`. Study the examples of panel functions below:

```
panel.straight.line <-
  function(x, y, col = par("col"), bg = NA, pch = par("pch"),
           cex = 1, col.smooth = "red")
  { datmat <- cbind(x, y)
    datmat <- datmat[order(datmat[, 1]), ]
```

```
    points(datmat[, 1], datmat[, 2], pch = pch, col = col, bg
    = bg, cex = cex)
    lines(datmat[, 1], fitted(lm(datmat[, 2] ~ datmat[, 1])),
    col=col.smooth)
  }

panel.poly <-
  function(x, y, col = par("col"), bg = NA, pch = par("pch"),
  cex = 1, col.smooth = "red")
  { datmat <- cbind(x, y)
    datmat <- datmat[order(datmat[, 1]), ]
    points(datmat[, 1], datmat[, 2], pch = pch, col = col, bg
    = bg, cex = cex)
    lines(datmat[, 1], fitted(lm(datmat[, 2] ~
     poly(datmat[, 1], 2))), col=col.smooth)
}

panel.smooth <-
  function (x, y, col = par("col"), bg = NA, pch = par("pch"),
  cex = 1, col.smooth = "red", span = 2/3, iter = 3, …)
  { points(x, y, pch = pch, col = col, bg = bg, cex = cex)
    ok <- is.finite(x) & is.finite(y)
    if (any(ok))
    lines(stats::lowess(x[ok], y[ok], f = span, iter = iter),
    col = col.smooth, …)
  }
```

11.15 Regression diagnostics

a) Study the help file of `lm.influence()` for information on its return values.

b) Standardized residuals as well as jackknife residuals where $stdres = \frac{\hat{e}_i}{s\sqrt{1-h_{ii}}}$ and $studres = \frac{y_i - \hat{y}_{(i)}}{\sqrt{var(y_i - \hat{y}_{(i)})}}$ can be obtained with the help of the following functions: `stdres()` and `studres()` available in package `MASS`. Study the help files of these functions.

c) Study how `predict()` and `fitted()` work.

d) Stepwise model selection: study the use of `step()`.

e) What is returned by the function `dfbetas()`?

f) Discuss the output of

```
> step(lm(MPG.highway ~ Length + Width + Weight + EngineSize
+ RPM + Rev.per.mile, data=Cars93))
```

g) Calculate and interpret the ***dfbetas*** associated with the object `lm.city.front` created in Section 11.7.

h) Continuing (g): obtain a graph with the `stdres` residuals on the y-axis and the fitted y-values on the x-axis and interpret the graph.

i) Continuing (g): obtain a histogram of the ***studres*** residuals. Interpret.

11.16 Experimental design

During experimental design it is often useful to predict the value of the dependent variable at every combination of the levels of the factor variables. Write an R function for this task that makes provision for any number of factor arguments and that also provides a dataframe with the factors as the columns and every combination of levels as the rows. Every levels-combination can only appear once. The function must be user friendly and must test if a given independent variable is a factor variable.
Hint: Study the help file of `expand.grid()`.

11.17 Consider the following data

	Test 1	Test 2	Test 3	Test 4
Group A:	10	15	30	12
Group B:	125	130	148	115

Plot the data of the two groups in the form of two profiles on the same set of axes.

Plot the data against Test 1, Test 2, Test 3 and Test 4 on the x-axis.
The scale of the data of Group A must appear on the y-axis on the left-hand side and that of Group B on the y-axis on the right-hand side. A detailed legend must be provided.

12 Analysis of Variance and Covariance with R

In this chapter the focus is on the analysis of variance and analysis of covariance with R. Only a very brief introduction to one-way and two-way analyses will be given.

12.1 One-way ANOVA models

Consider the data set `Poultry.data` (to be sourced as an R object using the link https://drive.google.com/file/d/19Fswno4IibmF0aBXzZbTDGgYgPQ_Ab_4/view?usp=sharing) on the production of poultry on three production units. This data set consists of numerical and factor (categorical) variables. Assume that a researcher is interested in the mean `NetWt` of poultry produced on the three production units. Assume further that the given sample values can be considered to form three representative random samples from the populations of all poultry harvested from the respective production units. `ProdUnit` is a categorical or factor variable having three *levels* viz. `Feathers`, `CookleDoo` and `Eggies`. These three levels have an influence or *effect* on the continuous or numeric dependent variable `NetWt`. The researcher would like to know if these effects are the same in the three specified populations. Analysis of variance decomposes the total variance in the dependent variable into a between-groups variance and a within-groups variance (or error variance). The underlying model can be written as

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

where Y_{ij} represents the value of the numeric dependent variable for observation j coming from population i; μ is an effect common to all observations; α_i represents an effect found only in population i and ε_{ij} is an effect (e.g. measurement error) that is specific to only observation (ij). It is assumed that the ε_{ij} is normally distributed with a mean of zero and a constant variance σ^2 (i.e. the variance is the same in each of the populations). The statistical models to be tested is

$$H_0: \alpha_1 = \alpha_2 = \cdots = \alpha_k$$

where k denotes the number of different populations, versus

$$H_A: At\ least\ one\ of\ the\ \alpha's\ differs\ from\ the\ rest$$

In general $\mu, \alpha_1, \alpha_2, \ldots, \alpha_k$ cannot all be estimated uniquely. In order for a statistical hypothesis to be ***testable*** it must be expressed in terms of ***estimable functions*** of the parameters. A statistical hypothesis can often be expressed in different equivalent forms. The null hypothesis above, for example, is equivalent to the following hypotheses:

$$H_0': \mu + \alpha_1 = \mu + \alpha_2 = \cdots = \mu + \alpha_k$$
$$H_0'': \alpha_1 - \alpha_k = \alpha_2 - \alpha_k = \cdots = \alpha_{k-1} - \alpha_k = 0$$

The latter two hypotheses are both in terms of $k - 1$ linearly independent estimable functions and hence are testable. The default behaviour of R is to estimate μ (the intercept) using the data, take (estimate) one of the α^s as zero and then estimate the remainder of the α^s as deviations from the estimated intercept using the data. Notice, that in doing so the given null hypothesis above can still be tested.

a) Use `str(Poultry.data)` to inspect the structure of the R object `Poultry.data`.

b) Use `with()` and `tapply()` to obtain the mean and the standard deviation of `NetWt` for the different `ProdUnits`.

c) Test if the effects of the three `ProdUnits` on `NetWt` are the same by executing the R command `lm(NetWt ~ ProdUnit, data = Poultry.data)` and using the commands `summary()` and `anova()`.

d) Use `coefficients()` and interpret the estimates referring to (ii).

e) Obtain the residuals and plot them against the fitted values. Use different colours for the different `ProdUnits`. Interpret the plot.

12.2 Two-way ANOVA models: Main effects and interaction effects

Inspection of the `Poultry` data by requesting `str(Poultry.data)` reveals the presence of a second factor variable `FeedMix` as well as a third factor variable `Year`. Notice that one of the feed mixtures has been introduced in 2012 for the first time.

Therefore, construct a subset of the data, `Poultry2012`, consisting only of the data collected in 2012.

We are interested in `NetWt` as a function of the two factor variables `ProdUnit` and `FeedMix` for the year 2012. In addition to the ***main effects*** of `ProdUnit` and `FeedMix`, respectively, we have also to consider ***interaction effects***.

Interaction can be defined as the result where one factor is modifying the effect of another factor.

In order to provide for a second factor together with interaction effects the two-way model is given by

$$Y_{ij} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ij}$$

In this model α represents the *main effect* of `ProdUnit` and β the *main effect* of `FeedMix` while the *interaction effect* is represented by $(\alpha\beta)$. The first hypothesis to be tested is the null hypothesis that there exist no interaction effects or that the interaction effects are negligible. Symbolically this is the hypothesis

$$H_0: (\alpha\beta)_{11} = (\alpha\beta)_{12} = \cdots = (\alpha\beta)_{mn}$$

where *m* denotes the number of levels in the first factor variable and *n* the number of levels in the second factor variable. In order to test for interaction we must have $r > 1$ repetitions (replicates) of each $(\alpha_i\beta_j)$ combination. The value of r in our example can be found using the following code:

```
> with(Poultry2012, table(FeedMix, ProdUnit)) ↲
```

resulting in

```
          ProdUnit
FeedMix    CookleDoo Eggies Feathers
  AB1             40     39       40
  TB5X            40     40       40
  X12             40     40       39
```

From the table above it is clear that not all treatment combinations occur exactly the same number of times. This is due to two missing values in the data. The correct solution to this problem is to estimate the missing values by the mean values of the available data in the affected cells. These values are then substituted in place of the missing values to form a complete data set. The ANOVA is then performed on the complete data set but the degrees of freedom for error is reduced with the number of missing values.

The presence of interaction can be investigated with an interaction plot using the following instruction with and without argument `xpd = FALSE`

```
> with(Poultry2012,interaction.plot(FeedMix,ProdUnit,  Response  =
  NetWt))
```

The result of the above instruction is given in Figure 12.2.1. What can be decuded from Figure 12.2.1? Explain in detail.
The formal ANOVA can be performed with the instruction

```
> two.way.anova<-lm(NetWt ~ ProdUnit + FeedMix + ProdUnit:FeedMix,
data=Poultry2012)
```

An analysis-of-variance table can be constructed with the instruction:

```
> anova(two.way.anova)
Analysis of Variance Table
Response: NetWt

                  Df  Sum Sq Mean Sq F value    Pr(>F)
ProdUnit           2  59.077  29.538 238.389 < 2.2e-16 ***
FeedMix            2 187.266  93.633 755.662 < 2.2e-16 ***
ProdUnit:FeedMix   4  16.315   4.079  32.918 < 2.2e-16 ***
Residuals        349  43.244   0.124
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1
```

It is clear from the above table that the null hypothesis of no interaction is to be rejected. This means that it would not make sense to test the main effects of `ProdUnit` and `FeedMix`. On the other hand, if no significant interaction would have been found then the F-values associated with the two main effects could have been investigated to conclude if none, one or both of the main effects are significant.

a) Repeat the above analysis using in turn each of the other numeric variables as the response variable.

b) Plot the residuals versus the fitted values and interpret the graphs. What can be concluded regarding the assumptions of the two-way ANOVA model?

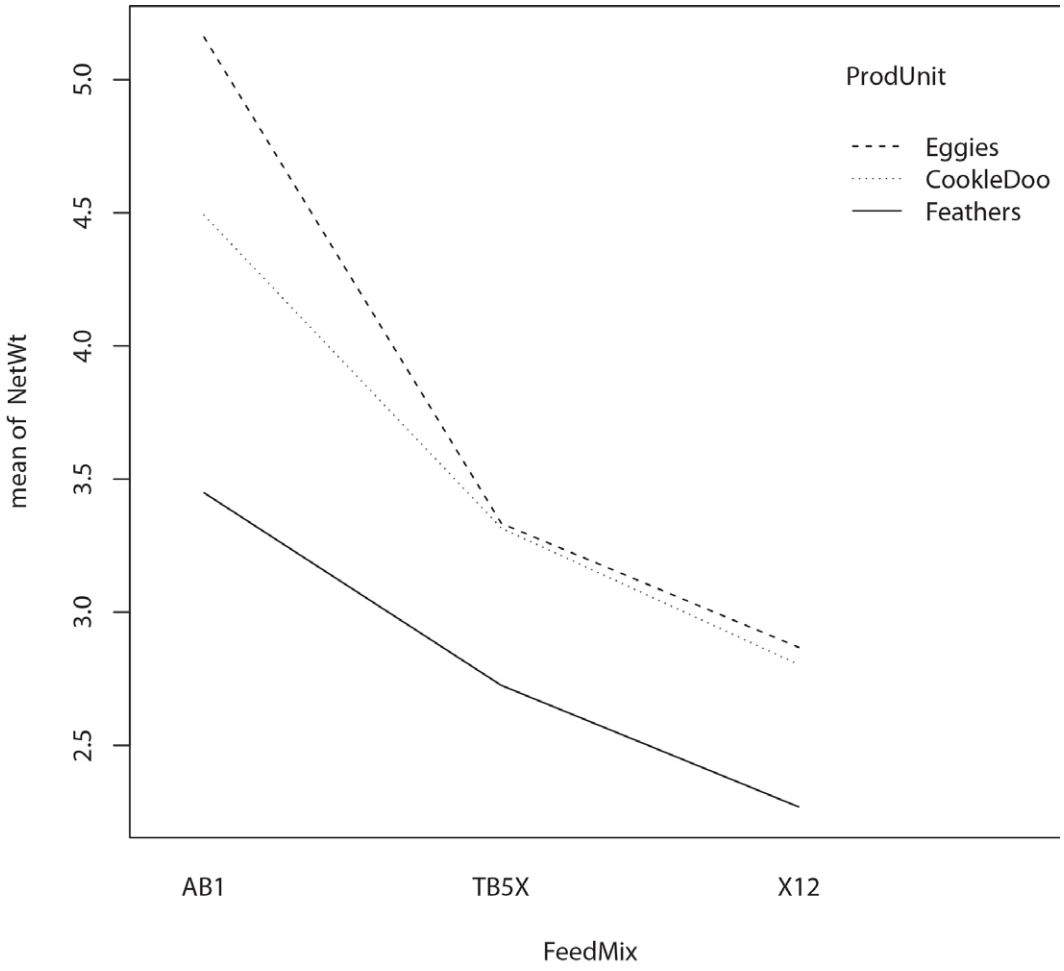


Figure 12.2.1: Interaction plot of factors `ProfUnit` and `FeedMix` associated with the dependent variable `NetWt`.

12.3 One-way ANCOVA models

A scatterplot of `NetWt` versus `BrutWt` suggests the possibility of a linear relationship between these two numeric variables. Check the above statement.

This means that the ANOVA model is not the correct model to use for the `Poultry` data. Instead we should consider a one-way ANCOVA model with `BrutWt` as a concomitant variable:

$$Y_{ij} = \mu + \alpha_i + \beta_i x_{ij} + \varepsilon_{ij}$$

where $i = 1, 2, \ldots, k;\ \ j = 1, 2, \ldots, n_i.$

In this model x_{ij} represents a continuous (or numeric) variable that is measured together with the dependent continuous variable Y_{ij}. If variable x has a significant relationship with the dependent (response) variable the values of the latter should be adjusted before performing a one-way ANOVA. The researcher must first ascertain whether the regression coefficients (i.e. the slopes of the regression lines) of the different groups are the same. Therefore, ***Step I*** of a covariance analysis investigates

$$H_{01}: \beta_1 = \beta_2 = \cdots = \beta_k$$

and if this is *not* the case, then the analysis proceeds by (i) estimating the regression parameters $\beta_1, \beta_2, \ldots, \beta_k$ for every group/treatment/level individually and (ii) by investigating the differences between the y-means of the k groups at *specific* values of x. On the other hand, if Step I shows that it can be assumed that $\beta_1 = \beta_2 = \cdots = \beta_k (= \beta)$, then the β_i in the model is replaced by the common β and Step II is carried out. ***Step II*** then consists of determining whether there is a significant regression relationship, *i.e.* the null hypothesis

$$H_{02}: \beta = 0$$

is tested. If it can be assumed that $\beta = 0$, then an ordinary analysis of variance can be carried out. If $\beta \neq 0$, *Step III* of ANCOVA is executed which first consists of adjusting the y_{ij}-observations for their regression relationship with the $\{x_{ij}\}$. An analysis of variance is then carried out on the adjusted $\{y_{ij}\}$ in order to ascertain whether the levels of the experimental factor have similar effects or not.

As an example we will again make use of the data for 2012 in the `Poultry` data set, i.e. the dataframe `Poultry2012` constructed above.

a) Obtain a scatterplot of `NetWt` as a function of `BrutWt`. Use different colours for the three production units. What can be deduced from the scatterplot?

b) Carry out a one-way ANOVA of `NetWt` obtained from the three production units in 2012.

c) Perform Step I of a one-way ANCOVA using the commands:

```
> one.way.ancova.I <- lm(NetWt ~ ProdUnit + BrutWt +
                       BrutWt:ProdUnit, data=Poultry2012)
> anova(one.way.ancova.I)
Analysis of Variance Table
Response: NetWt
                 Df  Sum Sq Mean Sq F value    Pr(>F)
ProdUnit          2  59.077  29.538  382.94 < 2.2e-16 ***
BrutWt            1 197.038 197.038 2554.39 < 2.2e-16 *** (B)
ProdUnit:BrutWt   2  22.636  11.318  146.72 < 2.2e-16 *** (A)
Residuals       352  27.152   0.077
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> coefficients(one.way.ancova.I)
   (Intercept)             ProdUnitEggies
   -0.6938633              -2.8970806
   ProdUnitFeathers        BrutWt
   -3.0866812              0.2843872
   ProdUnitEggies:BrutWt   ProdUnitFeathers:BrutWt
   0.2667621               0.2871409
```

12.3.1 Interpretation of the ANOVA table

First look at line (A). Line (A) provides information on whether the hypothesis $\beta_1 = \beta_2 = \beta_3$ should be rejected (as is the case here. Why?) If the conclusion would have been that all three β's are the same (differences are negligent) then (and only then) inspect line (B) to decide if the common slope $\beta = 0$. This is Step II. If it can be concluded that the common slope is zero then the ANOVA of (ii) is performed.

If in Step II it is concluded that the common slope is not equal to zero then perform Step III by calling, for example,

```
out.III <-lm(NetWt ~ BrutWt + ProdUnit, data=Poultry2012).
```

Note that in the latter call the numeric concomitant variable MUST PRECEDE the categorical variable on the right-hand side of the tilde because Step III is only carried out on condition that Step II has indicated a significant relationship between the concomitant variable and the response variable. The line corresponding to the factor variable in the analysis of variance table constructed with `anova(out.III)` is then inspected to conclude if the effects associated with the respective levels of the factor variable are significantly different.

i. In the example above we concluded that the three β's in the ANCOVA model are not the same. Obtain the estimated regression equations for the separate production units using the following instructions:

```
> one.way.ancova.sep.reglns <- lm(NetWt ~ -1 +
                    ProdUnit/BrutWt , data=Poultry2012)
> coefficients(one.way.ancova.sep.reglns)
```

i. Add the regression lines to the scatterplot constructed in (i). See Figure 12.3.1.

ii. Inspection of Figure 12.3.1 shows clearly that at `BrutWt` of 11.06 there is very little difference in the effects of the different `ProdUnits` on `NetWt`, but at a `BrutWt` of 16.0 the effect of `CookleDoo` on `NetWt` is quite different from the effects of the other two production units. Differences as these can be statistically investigated as follows:
First notice that the Residual sum of squares in the ANOVA table in (iii) is $RSSE = 27.152$ with 352 degrees of freedom. At `BrutWt` = 11.5 execute:

```
> anova(lm(NetWt ~ ProdUnit:(I(BrutWt - 11.06)) ,
         data=Poultry2012))
Analysis of Variance Table
Response: NetWt
                          Df  Sum Sq Mean Sq Fvalue   Pr(>F)
ProdUnit:I(BrutWt-11.06)  3 278.535  92.845   1201 <2.2e-16
Residuals                354  27.367   0.077
```

Now calculate $F_0 = \big((27.367 - 27.152) \div (354 - 352)\big)/(27.152 \div 352) = 1.3936$ and the corresponding p-value using the instruction

```
> 1- pf(1.3936, 2, 352)
[1] 0.2495
```

What is your conclusion?

iii. Repeat the calculations in (vi) at a `BrutWt` value of 15.6. What conclusions can be made?

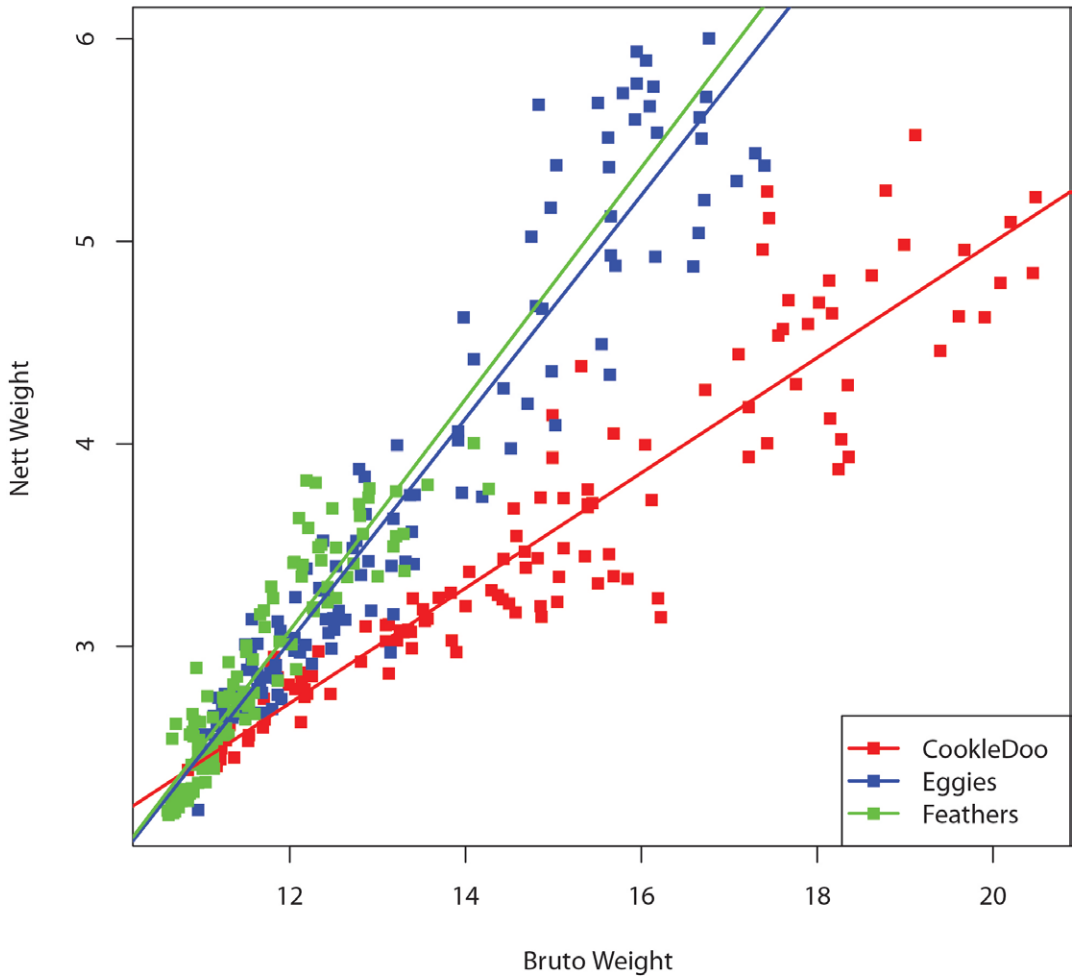


Figure 12.3.1: Scatterplot of `NetWt` vs `FeedMix` with separate regression lines for each of the three `ProdUnits`.

12.4 Maize example

A researcher investigates if three maize varieties give the same yield. The following data are obtained according to the requirements of a linear model:

a) Plot the given data as a single scatterplot that distinguishes between the three varieties.

b) What hypotheses and models are of importance when considering a covariance analysis of the given data?

c) Perform a detailed one-way ANCOVA. Obtain estimates for the regression coefficients in the regression equations

i. $Y_{ij} = \mu + \alpha_i + \beta x_{ij} + \varepsilon_{ij} \quad i = 1, 2, \ldots, k; \;\; j = 1, 2, \ldots, n_i$

ii. $Y_{ij} = \mu + \alpha_i + \beta_i x_{ij} + \varepsilon_{ij} \quad i = 1, 2, \ldots, k; \;\; j = 1, 2, \ldots, n_i$

Interpret the results of the analysis of covariance using the output of `lm()`.

VARIETY					
KB1		KB2		KB3	
Yield per hectare	Rainfall	Yield per hectare	Rainfall	Yield per hectare	Rainfall
17.50	95.90	37.50	114.10	18.75	98.00
45.00	116.20	41.25	119.28	13.75	117.60
38.75	116.20	48.75	121.94	52.50	105.28
33.75	93.10	32.50	97.02	17.50	94.50
18.75	81.34	53.75	102.90	18.75	92.40
21.25	115.36	38.75	91.70	52.50	110.74
		22.50	102.76	32.50	105.70
		17.50	78.54		

Table 12.4.1: Maize example data.

12.5 Heart rate example

Milliken and Johnson (2002) discuss an investigation by a sport physiologist of the effect of three exercise programs (EX1, EX2 and EX3) on heart rate (HR) of males between 28 and 35 years of age. The experiment consists of the random allocation of a subject to one of the programs. The subject then had to follow the exercise program for 8 weeks. On finishing the program each subject had to complete a 6 minutes run and then had his heart rate (EHR) measured. As a measure of their fitness before entering the experiment each subject had also his resting heart rate (AHR) determined prior to the experiment. The following data were obtained:

EXERCISE PROGRAM					
EX1		EX2		EX3	
EHR	AHR	EHR	AHR	EHR	AHR
118	56	148	60	153	56
138	59	159	62	150	58
142	62	162	65	158	61
147	68	157	66	152	64
160	71	169	73	160	72
166	76	164	75	154	75
165	83	179	84	155	82
171	87	177	88	164	86

Table 12.5.1: Heart rate example data.

Analyze the data and discuss the results.

13 Introduction to Optimization

Two related topics are briefly discussed in this chapter. First, the focus is on solving the equation $f(x) = 0$. The following two methods are considered (a) the bisection or binary search method and (b) the Newton-Raphson method. In the last two sections some principles underlying optimization are considered by examining briefly the usage of the function `optim()` for general optimization and the packages `lpSolve` and `Rsolnp` for constrained optimization.

13.1 The bisection method for solving $f(x) = 0$

The bisection method is based on the ***Mean Value Theorem*** (MVT): Let $f(x)$ be a continuous function defined over the interval $x = 0$ with the property that $f(a)$ and $f(b)$ have opposite signs. According to the MVT there exists a number p in (a, b) such that $f(p) = 0$. In order to keep arguments simple it is assumed that the root of the equation is unique in this interval. The bisection method consists of

i. repeatedly halving subintervals of $[a, b]$ and

ii. to determine in each step which half contains p.

The process is initialized by setting $a_1 = 1$ and $b_1 = b$. Let p_1 represents the midpoint of $[a_1, b_1]$, i.e. $p_1 = a_1 + \frac{b_1 - a_1}{2} = \frac{a_1 + b_1}{2}$. Suppose $f(p_1) = 0$ then $p = p_1$ so that we have the root. Suppose $f(p_1) \neq 0$ then $f(p_1)$ has similar sign than either $f(a_1)$ or $f(b_1)$. If $f(p_1)$ and $f(a_1)$ have similar signs then $p \in (p_1, b_1)$ and we set $a_2 = p_1$ and $b_2 = b_1$. If $f(p_1)$ and $f(a_1)$) have opposite signs then $p \in (a_1, p_1)$ and we set $a_2 = a_1$ and $b_2 = p_1$. The whole process is now repeated with the interval $[a_2, b_2]$. This process is illustrated in Figure 13.1.1 and coded in the R function, `Bisection()`, given below.

```
> Bisection <-
    function (a, b, fun, eps=.Machine$double.eps^0.4, maxiter=100)
{ # This function finds root of continuous function f(x) =
  # 0 in the interval [a, b] where
  # f(a) and f(b) have opposite signs.
  # The function f(.) is given in the form of an R
  # function in the argument fun

  Iter <- 1
  while (Iter <= maxiter)
     { fa <- fun(a)
       p <- a + (b-a)/2
       fp <- fun(p)
       if((fp == 0) | ((b-a)/2 < eps))
        { p <- p
          break }
       else
        { Iter <- Iter + 1
          if ( fa * fp > 0 )
             { a <- p
               fa <- fp }
          else b <- p
        }
     }

 if (Iter >= maxiter)
  stop("Process has not converged. Try increasing maxiter.\n")
 cat("Solution=: ", p, "\n")
}
```

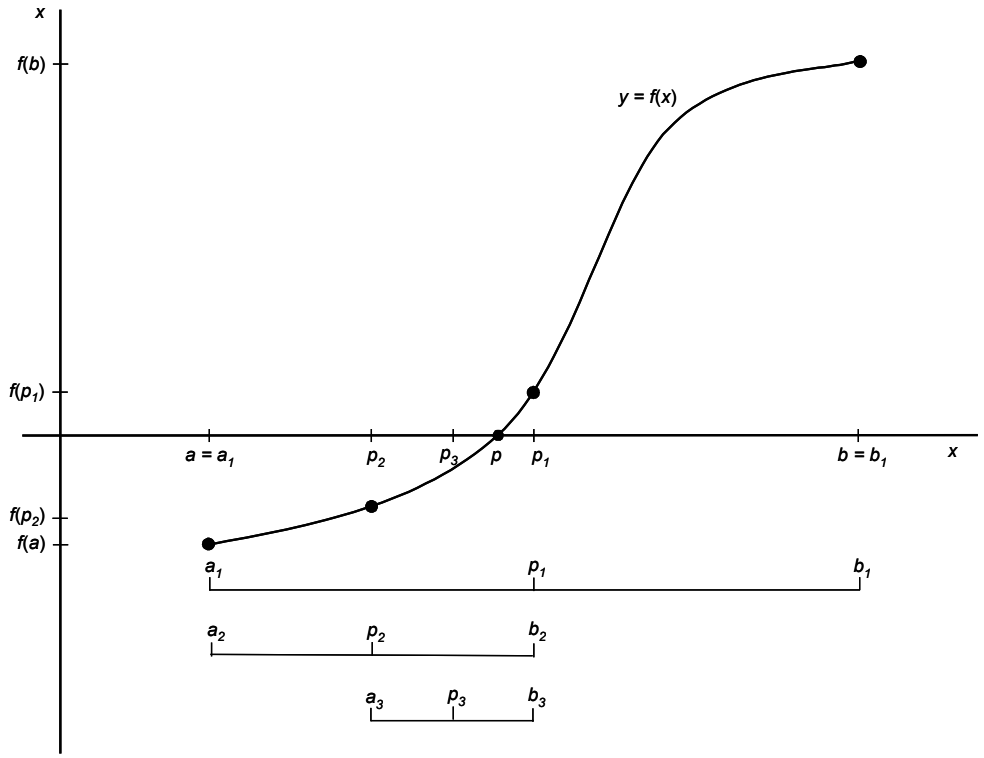


Figure 13.1.1: Principle underlying the bisection method for finding a root of $f(x) = 0$.

a) Use the bisection method to solve for any *given* c and n the equation

$$c^2 + x^2 + \frac{2cx}{n-1} = n - 2$$

The function must check whether $b > a$ and whether $f(a)$ and $f(b)$ have opposite signs.

b) Change the bisection function to provide for `fun` to accept several arguments. Demonstrate your function with the equation given in (a).

c) Consider the following data given by Dobson and Barnett (2008).

Season	1	2	3	4	5	6	7	8	9	10	11	12	13
Number of tropical cyclones	6	5	4	6	6	3	12	7	4	2	6	7	4

Table 13.1.1: Tropical cyclone data.

Suppose the number of cyclones can be modelled by a Poisson distribution with parameter θ.

i. Write down the log-likelihood function for determining the maximum likelihood (m.l.) estimate of θ.

ii. Use the function written in (b) without changing it to obtain the m.l. estimate of θ as well as the maximum of the log-likelihood.

iii. Use the R function `deriv()` to write a function based on the bisection principle for finding the maximum of $f(x, \theta)$ with respect to θ. Demonstrate its usage.

a) Study the help file of the R function `uniroot()`. Answer questions (a) and (c) using `uniroot()`.

13.2 The Newton-Raphson method

Consider the following: Let $f(x)$ be a function that is at least twice differentiable over the interval $[a, b]$. The value of $x = p$ such that $f(p) = 0$ needs to be found. Let $p_m \in [a, b]$be an approximate value of p such that $f'(p_m) \neq 0$ and $|p - p_m|$ is "small". Consider the Taylor power series expansion of $f(x)$ about p_m:

$$f(x) = f(p_m) + (x - p_m)f'(p_m) + \frac{(x - p_m)^2}{2} f''(\xi(x))$$

Where $\xi(x)$ lies between x and p_m. Since $f(p) = 0$, it follows from the previous equation that

$$0 = f(p) = f(p_m) + (p - p_m)f'(p_m) + \frac{(p - p_m)^2}{2} f''(\xi(p))$$

but $|p - p_m|$ "small" implies that $(p - p_m)^2$ is much smaller and therefore

$$0 \cong f(p_m) + (p - p_m)f'(p_m)$$

so that

$$p \cong p_m - \frac{f(p_m)}{f'(p_m)}.$$

From this follows the Newton-Raphson algorithm. It starts with an initial approximation p_0 and generates se sequence $\{p_m\}$ where

$$p_{m+1} \cong p_m - \frac{f(p_m)}{f'(p_m)}$$

for $m \geq 0$. In Figure 13.2.1 it is illustrated how the approximations are found by consecutive tangent lines.

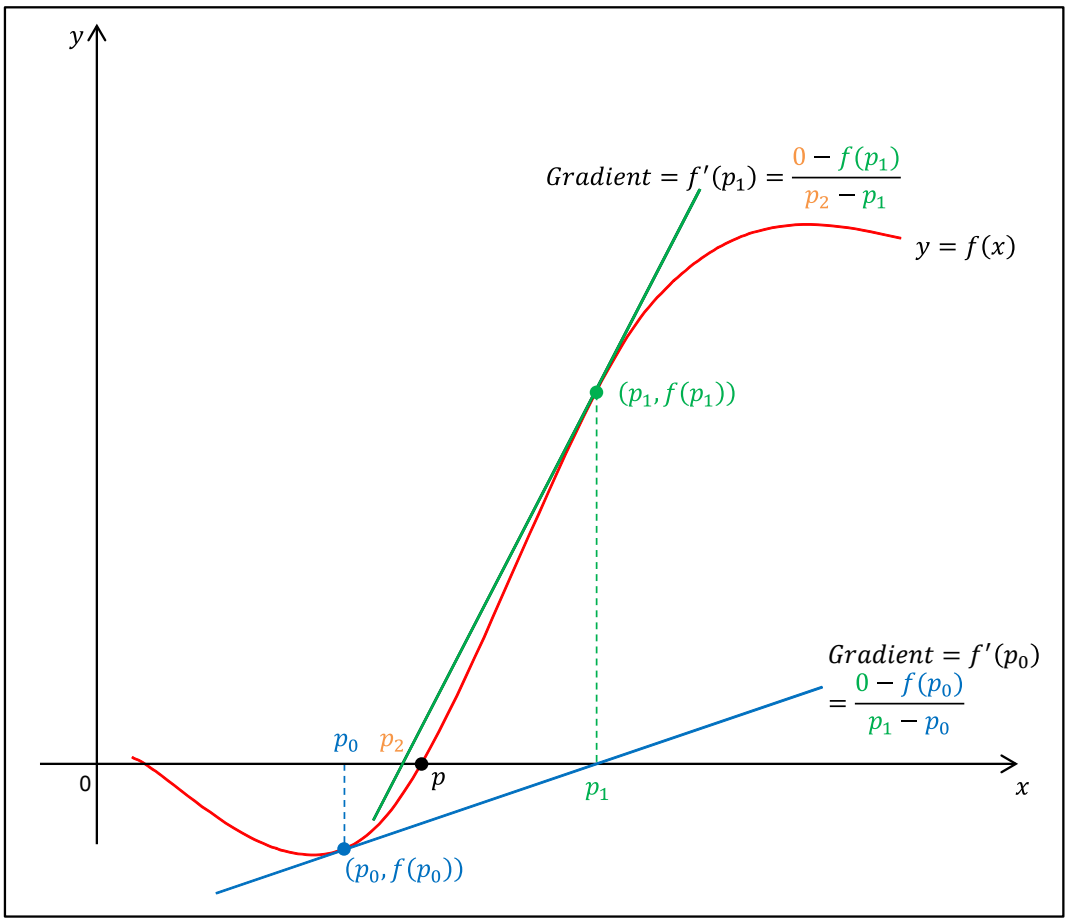


Figure 13.2.1: Principle underlying the Newton-Raphson algorithm.

a) When the Newton-Raphson algorithm is used for finding m.l. estimates the score statistic, U, and its derivative viz. U are needed. The statistic U can be approximated by $\mathrm{E}(U') = -\Im = -var(U)$. When U is substituted by $\mathrm{E}(U')$ in the Newton-Raphson algorithm the method is known as the ***(Fisher) scoring method***.

b) Write an R function to implement

   i. the Newton-Raphson algorithm and

   ii. the scoring method for finding numerically the m.l. estimate of the scale parameter of the Weibull-distribution

$$f(y_1, \ldots, y_n) = \prod_{i=1}^{n} \frac{\lambda y_i^{\lambda-1}}{\theta^{\lambda}} exp\left[-(y_i/\theta)^{\lambda}\right]$$

where $y_i > 0$. Experiment with various initial values and convergence criteria. The following function serves as an example:

```
NewtonRaphson <- function(fun1, fun2, initval, maxiter=20, eps=.
Machine$double.eps^0.4, …)
{ # initval = initial approximation of root
  # fun1 = function for which root is sought
  # fun2 = derivative of fun1 or the expected value of the
  # derivative of the score statistic

 count <- 1
 Iter <- count
 Outvec <- as.vector(initval)
 Estim <- initval – fun1(initval,…) /fun2(initval,…)
 while (abs (Estim – initval) > eps)
  { count <- count + 1
    Iter <- c(Iter, count)
    initval <- Estim
    Estim <- initval – fun1(initval,…)/fun2(initval,…)
    Outvec <- c(Outvec , Estim)
    if (count > maxiter)
     {   warning("Max number iterations reached without
     convergence")
         break
     }
  }
 data.frame(IterNo = Iter, Estim = Outvec)
}
```

The Weibull distribution is often used to model the time to failure (i.e. the survival time) of organisms or components. The parameter λ determines the shape while θ is a scale parameter. The following table (from Andrews and Herzberg (2008) Table 29.1) contains the lifetimes of a random sample of pressure vessels:

1051	1337	1389	1921	1942	2322	3629	4006	4012	4063
4921	5445	5620	5817	5905	5956	6068	6121	6473	7501
7886	8108	8546	8666	8831	9106	9711	9806	10205	10396
10861	11026	11214	11362	11604	11608	11745	11762	11895	12044
13520	13670	14110	14496	15395	16179	17092	17568	17568	

Table 13.2.1: Lifetime of pressure vessels data.

1. Use your implementation of the Newton-Raphson method to obtain the m.l. estimate of θ for given λ = 2.
2. Change your Newton-Raphson function so that the derivative is automatically calculated i.e. the function has only the arguments `fun`, `begin`, `eps`, `maxit` .

13.3 The R functions `optim()` and `constrOptim()`

a) In many cases closed form expressions do not exist for the maximum likelihood estimates of parameters. In such cases iterative procedures must be used to maximize the likelihood function. The functions `optim()` and `constrOptim()` in the package `stats` are useful in this regard. Study the help files of these two functions.

b) It follows from (a) that argument `par` of `optim()` contains the initial values for the parameters to be estimated; its second argument, `fn` is the function to be minimized. The first argument of function `fn` is a vector representing the parameters to be estimated while its second argument provides for the sample data for evaluating the function. Use the function `optim()` to find the maximum of the likelihood function as well as the maximum likelihood estimates for the parameters in the case of a random sample of size 100 from a $beta(4;\ 2)$-distribution. Investigate the role of different starting values as well as of changing the sample size.

c) Consider the `company.10var` data set introduced in Chapter 6.

   i. Center and scale the data.

   ii. Use the function `dist()` to obtain a distance matrix for the 79 companies.

   iii. What is the dimension of the space in which the distances between the 79 companies can be exactly represented? Let the exact distance between companies r and s be given by δ_{rs}.

iv. Suppose **X**: 79 × 2 is an (unknown) matrix giving the coordinates of the companies in a two dimensional display space such that the Euclidean distances $\{d_{rs}\}$ derived from **X** are the best approximations of the $\{\delta_{rs}\}$ in the sense that the following objective function is minimized:

$$Stress = \frac{1}{\sum_{r<s} \delta_{rs}} \sum_{r<s} \frac{(d_{rs} - \delta_{rs})^2}{\delta_{rs}}$$

Use `optim()` to find **X**.

v. Plot the 79 companies as points in two dimensions such that the inter-company distances $\{d_{rs}\}$ are the best approximations of the true distances $\{\delta_{rs}\}$ in the full space. Calculate the value of $Stress$. Use different colours for the successful and unsuccessful companies. Report on the results.

13.4 Packages `quadprog`, `lpSolve` and `Rsolnp` for constrained optimization

The R object below, `MCap.log.vec` (available using link https://drive.google.com/open?id=1JRFfVcwcssYa3opoL2mKHLhw4KPCl4Of), contains the observed log valuations of 25 companies listed in the Industrials sector of the Johannesburg Stock Exchange. The 16 columns of the object `Mmat.log` (available using link https://drive.google.com/open?id=1xufu4GfSTL3oPA5ZMoSkKLq-hgqmCxpZ) represent 16 estimates for the valuations in `MCap.log.vec`. These estimates are all based on different information extracted from the annual published statements of the 25 companies.

```
> MCap.log.vec
       1        2        3        4        5        6        7        8
 23.5595 18.6038 21.0041 20.6150 21.7530 23.4602 20.3834 21.1357
       9       10       11       12       13       14       15       16
 21.7674 21.4651 20.1827 19.0267 22.2079 20.3806 21.1867 24.0853
      17       18       19       20       21       22       23       24
 20.8409 19.8095 23.4439 19.3476 24.7185 23.3101 19.7218 20.4679
      25
19.1235
> Mmat.log
```

ID	mGP	mEBITDA	mEBIT	mPBT	mPAT	mHEPS	mTA	mBVE
1	22.5341	22.8253	23.0257	23.1746	23.2258	23.1413	22.7626	22.9724
2	17.6355	18.6887	18.9198	19.0659	19.1677	19.1989	17.8068	17.9911
3	20.5984	21.1805	21.4146	21.6320	21.4826	21.4366	20.9634	20.8307
4	19.9815	19.8664	20.0410	18.5122	18.6857	18.9232	20.9021	21.2157
5	23.8582	22.7109	22.7101	22.7930	22.7374	21.2652	22.4649	21.5055
6	23.3813	23.3177	22.8825	22.2893	21.8201	22.2827	23.4703	23.1356
7	19.8608	20.0454	20.3615	20.8393	20.7536	20.7441	19.3895	19.9748
8	20.6229	21.2387	21.3601	21.3583	21.3822	21.1617	20.9101	20.4476
9	20.0352	20.5844	20.7531	20.9056	20.9309	21.0162	20.1413	20.9660
10	20.7297	20.3086	20.6392	20.3584	20.2172	20.3396	20.4302	20.5470
11	18.9715	18.8002	19.4239	19.4995	19.4869	19.8422	19.4346	19.4584
12	18.7678	18.4171	18.8147	18.7262	18.6601	18.8824	18.4526	18.4530
13	23.3406	21.6917	22.0330	22.0571	22.0759	22.2133	21.8935	21.1329
14	19.6354	19.2579	19.9507	20.3487	20.2820	20.2140	19.2807	18.5055
15	20.3397	19.0839	19.4091	19.1365	19.0536	19.5981	19.9782	19.9120
16	25.1928	23.8137	23.9051	24.0599	23.9285	23.9265	23.6379	23.3451
17	20.8994	20.7705	21.0626	21.1761	21.4018	20.8815	21.2733	21.3650
18	19.5236	18.3761	18.6806	18.0314	17.6692	18.9733	18.5328	15.6584
19	22.6539	22.8574	23.0341	23.3115	23.4806	23.2662	22.8339	22.6702
20	19.7384	18.8903	19.1376	19.1166	19.2538	18.9127	18.5369	18.8594
21	23.4698	22.9203	23.3569	23.7893	23.9670	24.4697	24.1773	24.9260
22	21.4264	22.2333	22.4310	22.5071	22.4785	22.8326	22.0867	22.2313
23	19.8907	19.9128	20.2720	20.7166	20.6848	20.5868	19.4839	19.6007
24	20.4321	20.7015	20.7158	21.0456	21.0000	20.9480	20.2114	20.5490
25	20.4927	18.6022	19.1023	18.5975	18.5812	18.9202	19.1337	19.1733

```
       mIC      mT   mCgbO mNCIfOA mNCIfIA     mOD   mFCFF   mFCFE
 1 22.4042 22.9806 22.6176 22.5988 19.5369 23.3865 21.7410 22.1032
 2 17.5896 17.5615 18.7695 18.7690 19.1092 18.5833 18.9994 19.2577
 3 21.0192 21.1378 20.9567 20.7924 21.3134 18.5013 19.9004 20.6755
 4 20.9599 20.4464 20.1337 20.1372 17.4937 19.8537 19.7365 18.0123
 5 22.4177 22.8092 23.0456 22.8638 20.5325 23.4970 22.8111 22.5434
 6 23.5070 23.8685 23.4723 23.5470 24.0241 23.1911 24.0885 24.1309
 7 19.2089 19.3568 19.9435 19.8008 20.4167 21.0456 20.6852 20.7333
 8 20.6375 21.1542 21.1470 21.1160 20.3423 20.9676 19.4819 20.1215
 9 20.1747 19.9122 21.0913 21.4019 22.1873 20.8496 21.9058 22.1477
10 20.4182 20.7383 20.3210 20.0122 21.1186 16.7319 20.3296 21.7467
11 19.1628 19.4331 20.2059 20.4040 21.2214 19.1351 20.3541 20.6025
12 18.4490 18.8852 19.0838 19.0604 18.5860 18.8366 18.8194 18.6486
13 21.5532 21.8883 22.0332 22.3518 23.1484 21.8714 22.8554 23.1394
14 19.1596 19.3423 19.6655 18.8599 18.1438 21.0721 19.8464 19.1980
15 19.7835 20.5900 20.2939 20.4168 20.7556 20.2765 20.4759 20.5181
16 23.6817 23.2864 23.5203 23.1568 21.8995 23.5695 21.7981 19.7739
17 21.2809 21.5685 20.7624 20.8695 21.6530 18.4460 21.6758 21.7215
18 18.3573 19.9769 19.5108 19.3929 19.5588 20.0362 19.5394 19.4408
19 22.8597 22.9617 22.8379 22.8702 23.3134 23.4815 23.5430 23.5390
20 18.4538 18.6253 19.3195 19.4443 19.8450 17.1941 18.8484 19.0672
21 24.1418 22.4589 21.8882 22.5470 21.6008 24.7249 22.2471 22.2302
22 21.8876 21.9018 22.4118 22.2901 22.8535 22.5567 22.9812 23.1111
23 19.3244 19.8542 19.6644 19.4557 19.8549 20.7628 20.2305 20.1779
24 20.1792 19.5976 20.6651 20.6120 19.3651 20.8642 19.6763 18.9019
25 19.2137 19.2895 19.5349 19.2385 19.0088 19.6044 19.1820 18.7071
```

a) Obtain the correlation between `MCap.log.vec` and each of the 16 estimates. Discuss the results.

Our challenge is to form a single composite function of the 16 different estimates that is the best indicator of the valuations of 25 companies. The weights of this composite function are constrained to be non-negative and they must sum to one. The performance of the composite measure is judged according to the following criteria, each subjected to the constraints above:

Minimum sum of squared differences
Minimum of the sum of the absolute differences
Minimum median of the differences

i. Minimum of squared differences
The objective function to be minimized can be written in the form

$$\min_{\boldsymbol{b}} \left(-\boldsymbol{d}'\boldsymbol{b} + \tfrac{1}{2}\boldsymbol{b}'\boldsymbol{D}\boldsymbol{b}\right)$$

Subject to $\boldsymbol{A}'\boldsymbol{b} \geq 0$.

- Motivate in detail the form of the above objective function.
- Study the helpfile of the function `solve.QP()` in package `quadprog`.

The minimisation can be accomplished by the following instructions:

```
> H <- Mmat.log
> y <- MCap.log.vec
> Dmat <- t(H) %*% H
> Amat <- cbind(1,diag(16))
> bvec <- c(1, rep(0,16))
> solve.QP (Dmat, dvec, Amat, bvec, meq=1)
```

- Motivate the use of the above instructions.
- Discuss the solution obtained.

ii. Minimum of the sum of the absolute differences

The objective function can be written as

$$\min_{\boldsymbol{a}}(|y_1 - \boldsymbol{m}_1'\boldsymbol{a}| + |y_2 - \boldsymbol{m}_2'\boldsymbol{a}| + \cdots + |y_n - \boldsymbol{m}_n'\boldsymbol{a}|)$$

$$\text{subject to } \begin{cases} \sum_{j=1}^{p} a_j = 1 \\ a_j \geq 0 \; for \; all \; j \end{cases}$$

The following algorithm can be used.

Step 1: Replace every absolute term $|y_i - \boldsymbol{m}_i'\boldsymbol{a}|$ with a new variable u_i.

Step 2: Add the following two constraints for every new variable $u_i u_i$:

$$y_i - \boldsymbol{m}_i'\boldsymbol{a} \leq u_i$$

$$-y_i + \boldsymbol{m}_i'\boldsymbol{a} \leq u_i$$

Step 3: Minimize $\min_{\boldsymbol{u}}(u_1 + u_2 + \cdots u_n)$ subject to

$$\begin{cases} \sum_{j=1}^{p} a_j = 1 \\ a_j \geq 0 \; for \; all \; j \\ y_i - \boldsymbol{m}_i'\boldsymbol{a} \leq u_i \\ -y_i + \boldsymbol{m}_i'\boldsymbol{a} \leq u_i \end{cases} \Leftrightarrow \begin{cases} \sum_{j=1}^{p} a_j = 1 \\ a_j \geq 0 \; for \; all \; j \\ \boldsymbol{m}_i'\boldsymbol{a} + u_i \geq y_i \\ \boldsymbol{m}_i'\boldsymbol{a} - u_i \geq y_i \end{cases}$$

Note that variables must be non-negative. Let $\boldsymbol{y}: n \times 1 =$ MCap.log.vec > 0 and $\boldsymbol{M}: n \times p =$ Mmat.log > 0. Minimising the sum of the absolute differences is equivalent to

$$\min_{\boldsymbol{a},\boldsymbol{u}} \left(0a_1 + \cdots + 0a_p + u_1 + \cdots + u_n\right)$$

$$\text{subject to} \begin{cases} \sum_{j=1}^{p} a_j = 1 \\ a_j \geq 0 \; for \; all \; j \\ \boldsymbol{m}_i'\boldsymbol{a} + u_i \geq y_i \\ \boldsymbol{m}_i'\boldsymbol{a} - u_i \geq y_i \end{cases}$$

Study the helpfile of the function `lp()` in the package `lpSolve`. The code below illustrates the usage of `lp()`.

```
> objective.in <- c (rep (0, ncol(Mmat.log)), rep (1,
nrow(Mmat.log)))
> const.dir <- c("=", rep(">=", ncol(Mmat.log) +
nrow(Mmat.log)), rep("<=", nrow(Mmat.log)))
> const.rhs.log <- c(1, rep(0, ncol(Mmat.log)),MCap.log.vec,
MCap.log.vec)
> temp1 <- c(rep(1, ncol(Mmat.log)), rep(0,nrow(Mmat.log)))
> temp2 <- cbind(diag(ncol(Mmat.log)), matrix(0, nrow =
ncol(Mmat.log), ncol = nrow(Mmat.log)))
> temp3 <- cbind(Mmat.log, diag(nrow(Mmat.log)))
> temp4 <- cbind(Mmat.log, -diag(nrow(Mmat.log)))
> Cmat <- rbind(temp1, temp2,temp3,temp4)
> out <- lp("min", objective.in, Cmat, const.dir, const.rhs.log)
```

`out$solution` is a vector of $p + n$ values. The first p are the elements of $\boldsymbol{a}$ that sum to one and are all non-negative.

`out$objval` is the minimum of the objective function attained by the solution above.

Use `str(out)` to find the solution. Discuss the solution found.

A standardized version of the objective function

$$\min_{\boldsymbol{a}}(|y_1 - \boldsymbol{m}_1'\boldsymbol{a}| + |y_2 - \boldsymbol{m}_2'\boldsymbol{a}| + \cdots + |y_n - \boldsymbol{m}_n'\boldsymbol{a}|)$$

is the function

$$\min_{\boldsymbol{a}}\left(\frac{|y_1 - \boldsymbol{m}_1'\boldsymbol{a}|}{y_1} + \frac{|y_2 - \boldsymbol{m}_2'\boldsymbol{a}|}{y_2} + \cdots + \frac{|y_n - \boldsymbol{m}_n'\boldsymbol{a}|}{y_n}\right)$$
$$= \min_{\boldsymbol{a}}\left(\left|1 - \frac{\boldsymbol{m}_1'\boldsymbol{a}}{y_1}\right| + \left|1 - \frac{\boldsymbol{m}_2'\boldsymbol{a}}{y_2}\right| + \cdots + \left|1 - \frac{\boldsymbol{m}_n'\boldsymbol{a}}{y_n}\right|\right)$$

for all y_i positive subject to $\begin{cases} \sum_{j=1}^{p} a_j = 1 \\ a_j \geq 0 \; for \; all \; j \end{cases}$

The following modifications are now needed:

Step 1: Replace every absolute term $|1 - \boldsymbol{m}_i'\boldsymbol{a}/y_i|$ with a new variable u_i.

Step 2: Add the following two constraints for every new variable u_i:

$$1 - \boldsymbol{m}_i'\boldsymbol{a}/y_i \leq u_i$$
$$-1 + \boldsymbol{m}_i'\boldsymbol{a}/y_i \leq u_i$$

Step 3: Minimize $\min_{\boldsymbol{u}}(u_1 + u_2 + \cdots u_n)$ subject to

$$\begin{cases} \sum_{j=1}^{p} a_j = 1 \\ a_j \geq 0 \; for \; all \; j \\ 1 - \boldsymbol{m}_i'\boldsymbol{a}/y_i \leq u_i \\ -1 + \boldsymbol{m}_i'\boldsymbol{a}/y_i \leq u_i \end{cases} \Leftrightarrow \begin{cases} \sum_{j=1}^{p} a_j = 1 \\ a_j \geq 0 \; for \; all \; j \\ \boldsymbol{m}_i'\boldsymbol{a}/y_i + u_i \geq 1 \\ \boldsymbol{m}_i'\boldsymbol{a}/y_i - u_i \geq 1 \end{cases}$$

As an example, consider the following R instructions:

```
> y.vec <- MCap.log.vec
> M.star <- sweep(Mmat.log, 1, y.vec, "/")
> n <- nrow(M.star)
> p <- ncol(M.star)
> temp1 <- c(rep(1, p), rep(0, n))
> temp2 <- cbind(diag(p), matrix(0, nrow=p, ncol=n))
> temp3 <- cbind(M.star, diag(n))
> temp4 <- cbind(M.star, -diag(n))
> Cmat <- rbind(temp1, temp2, temp3, temp4)
> const.dir <- c("=", rep(">=",p+n), rep("<=",n))
> const.rhs <- c(1, rep(0, p),rep(1, n), rep(1,n))
> objective.in <- c(rep(0, p), rep(1, n))
> out <- lp(direction="min", objective.in,
          const.mat=Cmat,const.dir, const.rhs)
```

Discuss the solution by comparing it to solution found for the non-standardized objective function.

iii. Minimum median of the differences

In general this minimization requires starting values for the iterative scheme. The choice of starting values may lead to a local optimum so that the global optimum is not found. One approach is to repeat the iterative algorithm for many random starts and to choose the best result from these.

Study the help file of function `gosolnp()` in package `Rsolnp`. Note how to specify the random starts. The functions `MinMed()` and `MinMed3()` implement `gosolnp()` for finding the solution for finding the minimum median of the differences.

```
> MinMed <- function (yvec,xmat,restarts=1)
{ if(length(yvec) != nrow(xmat))
    stop("Number of elements of yvec must equal number of rows
    of xmat \n")
  p <- ncol(xmat)
  fun1 <- function(a, y, mat) median(abs(y-mat %*% a))
  # fun1: objective function that will be minimized
  eqff <- function(a,y, mat) sum(a)
  eqBB <- 1
  # eqff and eqBB specify the equality constraint
  ineqff <- function(a,y,mat) a
  ineqLBB <- rep(0, p)
  ineqUBB <- rep(1, p)
  # ineqff, ineqLBB and ineqUBB specify the inequality
  # constraints

  # function gosolnp of package Rsolnp used for executing
  # the optimization
  gosolnp(fun = fun1, eqfun = eqff, eqB = eqBB, ineqfun =
          ineqff, ineqLB = ineqLBB, ineqUB = ineqUBB, LB =
          rep(0,p), UB = rep(1,p),
 n.restarts = restarts, y = yvec, mat = xmat)
}
> MinMed3 <- function (yvec,xmat,startvals = rep(1/ncol(xmat),
ncol(xmat)), control=list(tol = 1.0e-7,delta=1.0e-8))
{ # Similar to function MinMed but objective function in
  # standardized form
  # and with specified starting values i.e. no random starts
```

```
    #The default startvalues is equal probs.
    if(length(yvec) != nrow(xmat))
    stop("Number of elements of yvec must equal number of rows
    of xmat \n")
    p <- ncol(xmat)
    fun1 <- function(a, y, mat) median(abs(y-mat %*% a)/y)
    # fun1: objective function that will be minimized
    eqff <- function(a,y, mat) sum(a)
    eqBB <- 1
    # eqff and eqBB specify the equality constraint
    ineqff <- function(a,y,mat) a
    ineqLBB <- rep(0, p)
    ineqUBB <- rep(1, p)
    # ineqff, ineqLBB and ineqUBB specify the inequality
    # constraints

    # function gosolnp of package Rsolnp used for executing
    # the optimization

   solnp(pars = startvals,fun = fun1, eqfun = eqff, eqB =
        eqBB, ineqfun = ineqff, ineqLB = ineqLBB, ineqUB =
        ineqUBB, LB = rep(0,p), UB = rep(1,p), control =
        control,y = yvec, mat = xmat)
}
```

b) Call `MinMed()` with arguments `yvec = MCap.log.vec`, and `xmat = Mmat.log`, using 25 random starts.

c) Call `MinMed3()` with arguments `yvec = MCap.log.vec`, and `xmat = Mmat.log`, using the default start values. Repeat the call using as start values the result of the last call to `lp (direction = "min", objective.in, const.mat = Cmat,const.dir, const.rhs)` above.

References and Further Reading

Andrews, D.F. and Herzberg, A.M. (1985). *Data*, New York: Springer-Verlag. ISBN 978-1-4612-9563-1.

Becker, R.A., Chambers, J.M. and Wilks, A.R. (1988). *The New S Language: A Programming Environment for Data Analysis and Graphics*. Pacific Grove, CA: Wadsworth & Brooks/Cole.ISBN 0-534-09192-X.

Chambers, John M. (1998). *Programming with data: a guide to the S language*. Berlin: Springer. ISBN 0-387-98503-4.

Chambers, John M. (2008). *Software for data analysis programming with R*. Berlin: Springer. ISBN 0-378-75935-2.

Chambers, J.M. and Hastie, T.J. (Eds.) (1993). *Statistical Models in S*. Pacific Grove, CA: Wadsworth & Brooks/Cole. p. 624. ISBN 0-412-05291-1.

Dobson, A.J. and Barnett, A.G. (2008). *An Introduction to Generalized Linear Models*. Boca Raton, FL: CRC Press. ISBN 978-1-58488-950-2.

Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**(3):299-314.

Milliken, G.A. and Johnson, D.E. (2002). *Analysis of messy Data. Volume III: Analysis of Covariance.* New York: Chapman & Hall/CRC. ISBN 978-1584880837.

Spector, P. (1994). *An Introduction to S and S-Plus.* Pacific Grove, CA: Duxbury Press. ISBN 978-1-4398-3176-2.

Further reading

Crawley, M.J. (2013). *The R Book*. Chichester: Wiley. ISBN 978-0-470-97392-9.

For graphics, and in particular grid graphics
Murrell, P. (2011). R Graphics, 2nd edition. Boca Raton, FL: CRC Press.

Endnotes

1. `read.S()` can read the binary files produced in some older versions of S-PLUS on either Windows (versions 3.x, 4.x, 2000) or Unix (version 3.x with 4 byte integers).
2. This function requires SAS to be installed since it creates and run a SAS program that converts the data set to ssd format and uses `read.xport()` to obtain a dataframe.

Index

Symbols

: 15, 42, 50, 53, 54, 56, 57, 58, 59, 68, 69, 73, 89, 113, 127
:: 59, 121, 187, 188, 193
::: 59, 121
!= 41, 72, 74, 119, 129, 130, 220, 221
. . . 24, 117, 118, 119, 124, 142, 148,193, 211
( ) 26, 73
[ ] 26, 61, 73, 74, 104, 106
[[ ]] 61, 74, 104, 106
{ } 26, 73
@ 43, 44, 45, 143
# 23, 41, 42, 44, 45, 49, 50, 75, 93, 99, 114, 121, 124,127, 131, 140, 145, 146, 153, 164, 173, 177, 184,185, 186, 207, 211, 220, 221
<- 15, 16, 23, 32, 40, 41, 42, 44, 45, 50, 53, 54, 55, 56, 73, 74, 75, 78, 127
<<- 73, 74, 122
-> 23, 50
= 15, 23, 27, 28, 40, 43, 50, 52, 56, 61, 72, 74
== 23, 50, 93, 94, 124, 145, 147, 148, 157, 184, 207
$ 26, 43, 44, 53, 59, 61, 64, 74, 104, 105, 106, 153, 104, 106, 153
.C 129
.First 36, 40, 41, 42, 123
.Fortran 129, 130
.GlobalEnv 22, 24, 33, 57, 58, 61
.Last 40, 123
.Machine 24, 40, 71
.RData 16, 22, 33, 34, 36, 37, 39, 48, 49, 50, 55, 149, 154
.Rhistory 33, 34, 154

A

abline 114, 163
abs 75, 211, 220, 221
absolute value 85
acf 117, 142
acos, acosh 75
actual argument 75, 123
additive 181, 87
additive effects 181
additive model 187
add-on packages 11
adj 170
adjusted R-squared 184, 185
AIC 185
akima 90, 178
algorithm 210, 211, 216, 220
all 16, 27, 50, 55, 132, 147
all.names 55
along 58, 92, 93, 121
alpha.3d 93
alphabet 116
amsfonts 43
analysis-of-variance 198
ANCOVA 199, 201, 202, 204
Animation 177
anova 182, 183, 186, 187, 196, 198, 201, 202, 203
any 71, 132, 150, 151, 193
aov 180, 183, 186
aplpack 91
append 80, 110, 155
apply 42, 132, 133, 135, 139, 140, 148, 156, 157
apropos 121
arc 75
arg 75, 129
args 74, 95, 119
argsAnywhere 95
arguments 16, 23, 25, 50, 55, 59, 68, 73, 74, 75, 124
Arial 21
arithmetic 24, 68
array 98, 99
arrows 154, 163, 173, 174, 177
as.character 166
as.data.frame 141, 142, 186
as.integer 130, 142

as.matrix 62, 65, 109, 147
as.vector 211
as.xxx 62
ask 134, 160
asp 114, 116, 117, 177
aspect ratio 43, 114
assign 23, 50, 78, 79, 146
assigned 25, 55, 119, 122, 123, 146
assigning 23, 50, 78
Assignment 50, 79 assigns 23
asterisked 95, 117, 118, 142
asymptotic regression 189
atan, atanh 75
attach, attached 27, 37, 40, 42, 58, 90, 121, 177
attaching databases 61
attr 58, 63, 151
attribute 142, 182
attributes 53, 63, 180
Autoloads 57, 58, 61
axes 93, 94, 95, 117, 118, 168, 170, 171
axes are geometrically accurate 114
axis 45, 163, 168, 169, 170, 178, 194
a-z 26
a-zA-Z 26

B

backslash 18, 24, 26, 51
backspace 18, 48
backtick 24
bar chart 171, 172
barplot 88, 119
base 14, 53, 57, 58, 59, 61, 75, 127
begin 43, 44, 45, 52, 212
begin a session 34
beginning 26, 40, 155
bell 18
Bernoulli 66
beta 24, 94
between-groups variance 195
bf.maxit 187
bg 172, 192, 193
binary 24, 71, 154, 206, 223
binding 122

binomial 66
bisection 206, 208, 209
bit 14, 15, 71, 130
bivariate 95, 112, 174
body of function 74, 119, 122, 123, 129, 142
bootstrap 67
border 95
bottom 19, 36, 37, 73, 89, 116
box 95, 110, 163
boxplot 44, 45, 88, 89, 90, 111, 119, 120, 148
braces 24, 43, 73, 74, 119
brackets 26, 73, 104, 106
break 94, 120, 135, 139, 207, 211
break point 84, 120
browse 149
bs 188
B-spline 188
buffered output 20
by 45, 50, 114, 177
byrow 62, 99, 101, 104, 110
byte 71, 223
bytecode 117, 142

C

c 18, 44, 45, 50, 51, 52, 53, 55, 61, 62, 104, 109, 129, 147, 158, 176
call 57, 59, 119, 141, 145
caption 45
cargrp 164, 165, 166
Cartesian 114, 115
case sensitive 23
cat 18, 41, 138, 145, 147, 148, 149, 155, 156, 157, 207
categorical variable 53, 54, 55, 83, 84, 135, 202
cbind 76, 78, 110, 133, 173, 192, 193, 216, 217, 219
ceiling 76
center 77, 212
centring 77
cex 45, 166, 170, 177, 192, 193
character 18, 22, 25, 26, 50, 61, 62, 63, 78, 83, 84, 85, 86, 95, 105, 109, 125, 126, 127, 138, 155, 157, 166, 168, 169
Chernoff 91
chi-squared 139
chol 77
Cholesky decomposition 77
chunk 43, 44, 74
circle 114, 115, 116, 174, 175
class 11, 12, 53, 55, 63, 64, 65, 82, 107, 112, 142, 143, 151, 160, 180, 181, 182
classes 11, 22, 25, 26, 122, 143
clipboard 43, 56, 85, 153
cluster 92
CMD 130
code 18, 29, 43, 44, 45, 61, 71, 74, 94, 111, 113, 117
codes 54, 83, 128
coef 182
coefficient 81, 122, 133, 174, 181
coefficients 196, 200, 201, 202
col 91, 93, 95, 104, 121, 124, 160, 163, 164, 172, 192, 193
collapse 84, 85
colnames 62, 76, 110, 127, 156, 157
colon 18, 24, 50, 59
combinations 24, 198
combine 109
combining 71
Commander 38, 39
command line editing 48
commands window 16, 19, 29, 84, 93, 154
comment 23, 50, 74
communicating with the operating system 158
communication with graphs 92
communication with the R evaluator 18
company.10var 113, 212
comparison 23, 24, 50
compiled 17, 129
compiler 17
complex 63, 75, 175
computations by a computer 40
COMSPEC 158
concatenating 76
condition 98, 121, 135, 139, 151, 202
conditional computations 135
conditional inverse 86

conflicts 52, 59, 121
Conj 75
console 16, 18, 19, 20, 21, 23, 24, 28, 29, 30, 31, 32, 36, 37, 50, 123, 154, 155, 156, 158
constrained optimization 206, 213
constrOptim 212
contour 90
contributed packages 15, 21
control 24, 33, 79, 139
control argument 181
control of the flow 24
control operators && and || 138
cooks.distance.lm 182
coplot 90, 189, 192
copy 19, 39, 43, 56, 153
cos 91, 114, 177
Courier New 42
CPU 140
CRAN 21, 36
crossprod 77
csv 55, 153
cursor 18, 93
cut 83, 88, 91, 92, 133, 134, 169
cyclically 155

D

D 85
dash 26
database 16, 25, 55, 61, 78, 154
dataframe 62, 88, 89, 107, 108, 109, 110, 143, 180, 181, 194, 201, 223
dataframes 58, 97, 107, 108, 109, 132, 134
data set 28, 42, 72, 81, 82, 83, 88, 91, 92, 110, 111, 113, 121, 133, 134, 135, 144, 156, 158, 160, 164, 171, 178, 183, 186, 187, 188, 189, 192, 195, 198, 201, 212, 223
datasets 40, 55, 57, 58, 61
date 11, 121, 155, 158
dBase 154
dbf 153
debug 149
debugger 149
debugging 111
decimal 70
decimals 156
default 14, 24, 25, 27, 28, 31, 50, 55, 58, 77, 89, 95, 106, 114, 116, 117, 119, 120, 129, 142, 145, 155, 196, 221
default value 114, 116, 129, 145
degrees of freedom 184, 185, 198, 203
dendrogram 92, 117, 142
density function 82, 94, 171, 174
deparse 95, 125, 126, 127, 146
dependent variable 179, 186, 194, 195, 199
deriv 209
derivative 211, 212
derive 85
DescribeData 146, 148
det 78, 79
detach 57, 58
detail 52, 59
dev.ask 175
dev.copy 175
dev.cur 175
dev.list 175
dev.new 87, 164, 189
dev.next 175
dev.off 87, 175
dev.prev 175
dev.set 175
deviance 182
dfbeta 182
dfbetas 182, 193, 194
diag 77, 110, 127, 216, 217, 219
diff 52
differentiation 85
digits 45, 156
dim 53, 54, 55, 62, 65, 76, 98, 110, 127, 146
dimnames 62, 76, 98, 110, 127, 157
dir 35
direction 219, 221
directories 33, 35
directory 16, 22, 23, 33, 34, 36, 37, 41, 49, 55, 73, 74, 78, 79, 123, 131
dist 92, 212

distribution 15, 44, 66, 67, 82, 85, 92, 111, 112, 174, 208, 211, 212
division 68, 74
do.call 119, 141
dll 130, 131
dos 130, 158
dotchart 91
dotsMethods 24
double 18, 23, 24, 26, 40, 50, 59, 71, 104, 130, 207, 211
double-colon 59
draw 116, 174, 192
drop 100
drop1 182, 183, 185
dump 149, 154
duplicated 34, 80, 144
dyn.load 131
dynamic 129, 130
dynamic loading 129

E

ecdf 92, 117, 142
echo 44, 154
Edit 19
edit 21, 33, 42, 43
editor 20, 28, 31, 40, 154
effects 160, 180, 181, 182, 186, 195, 196, 197, 198, 201, 202, 203
eigen 67, 77
eigenvalues 67, 77, 116
eigenvectors 77, 116
elapsed 140
element 45, 71, 73, 75, 77, 80, 85
elements 61, 62, 63, 67, 77, 98, 99, 101, 103, 104, 107, 109, 110, 121, 138, 139, 140, 145, 148, 174, 217, 220, 221
elementwise 72, 140
ellipse, ellipsoid 116
else 41, 94, 119, 121, 123, 133, 135, 138, 145, 147, 151, 207
emph 43
empty 50, 56, 153
empty cells in Excel 153
empty object 63

encode 84
end keys 48
enter 15, 16, 31, 49
envir 78
environment 22, 23, 25, 28, 33, 36, 38, 39, 40, 50, 56, 57, 58, 59, 71, 79, 117, 121, 122, 123, 142, 146, 181
environmentName 58
Epi 154
EpiData 154
epiinfo 153
eps 71, 207, 211, 212
equality sign 23, 50
equality symbol 23
equal to 41, 140, 165, 202
error attr 58, 63, 151
Error handling 150
error message 51, 95, 106, 123, 150
Error tracing 149
Error: unexpected 50, 51
Esc 15
escape sequence 24
estimable function 181
Euclidean distance 127
eval 52, 95, 96, 126, 127, 142
evaluated 17, 59, 119, 127, 129
evaluation 122, 129
evaluator 17, 18, 23, 32, 58, 121
Excel 56, 153
exists 59, 146, 147, 158
exit 123, 139, 148, 149, 164, 165, 169, 176
exp 70, 75, 95
expand.grid 178, 194
explicit loops 140
exponentiation 71
export files 154
expression 25, 26, 52, 59, 101, 116, 122, 126, 127, 135, 138, 139, 142, 174, 181
external file 62, 154
external representation 154
external routines 129
extract 24 77
extractAIC 182

F

.First 36, 40, 41, 42, 123
f 18
F 52, 82, 112, 147, 148, 170, 184, 186
factor 53, 83, 91, 117, 118, 119, 142, 171, 180, 181, 187, 194, 195, 196, 197, 201, 202
factors 107, 180, 181, 194, 199
FALSE 44, 55, 60, 61, 62, 65, 71, 73, 77, 97, 100, 101, 108, 109, 120, 135, 198
family 11, 154, 182, 187
F-distribution 82
figure region 87
file 19, 20, 28, 35, 36, 39, 40, 49, 56, 62, 78, 85, 93, 147, 154
filename 18, 20, 40, 153, 154
fill 155, 156, 157
fin 159
find 52, 53, 58, 121
finds data 56
fine-tune plots 168
fine-tuning graphs 43
finite 193
fit 187, 188, 189
fitted 181, 184, 185, 189, 193, 194, 196, 199
fivenum 111
five-point summary 111
fix 28, 31, 32, 85, 120, 121, 122, 127, 129
floating point 40, 71
floor 76
flow statement 135
folder 33, 34, 36, 37, 39, 43, 46, 48, 49, 50, 55, 154
font 21, 42
foreign 153
for loops 138, 140
formal argument 75, 122, 123, 124
formal parameters 122
format 45, 147, 154, 156, 157, 223
formatting 153, 154, 155, 156, 157, 158
form feed 18
form submatrices 99, 110
formula 117, 142, 181, 182, 184
Fortran 129, 130, 131
FoxPro 154

frame 56, 146, 149
free variable 122
from 50, 56, 91, 114, 177
full2resp 144
full path 28, 31, 62, 131, 153
fun 122, 123, 207, 208, 212, 220, 221
function 16, 18, 21, 22, 25, 28, 29, 30, 31, 32, 33, 40, 41, 42, 50, 55, 56, 59, 61, 62, 63, 66, 67, 73, 74, 75, 77, 78, 79, 80, 81, 82, 86, 88, 89, 90, 91, 92, 93, 94, 95, 98, 99, 104, 109, 110, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 138, 139, 140, 141, 142, 143, 144, 145, 146, 148, 149, 150, 151, 152, 154, 155, 156, 158, 159, 161, 163, 164, 165, 168, 169, 171, 172, 173, 174, 175, 176, 177, 178, 180, 181, 182, 183, 186, 187, 188, 189, 190, 192, 193, 194, 196, 201, 206, 207, 208, 209, 211, 212, 213, 215, 216, 217, 218, 220, 221, 223
function in R 16, 74

G

gam 121, 187, 188
gamma 24, 75, 79
generalized 86, 187
generalized additive 187
generalized additive model 187
generalized linear models 187
generic function 142, 143
get 61, 95, 111, 178
getAnywhere 59, 95, 118, 127, 183
getenv 158
getwd 33, 49, 50
ggplot 159
glm 187
GlobalEnv 22, 24, 33, 57, 58, 61
Global environment 22, 23, 25
go 18
gosolnp 220, 221

graph 16, 37, 43, 66, 87, 88, 89, 91, 93, 94, 95, 112, 114, 115, 116, 123, 124, 146, 147, 160, 161, 163, 164, 165, 166, 171, 174, 175, 177, 186, 187, 191, 194
Graphical arguments 192
graphical parameter 165
graphical user interface 38, 39
graphic devices 87
graphics 11, 22, 42, 57, 58, 61, 87, 93, 114, 117, 118, 142, 147, 148, 159, 160, 164, 165, 174, 175, 176, 177, 222
graphics.off 87, 164, 165, 175
graph window 16, 43, 87, 89, 112, 115, 116, 123, 161, 166
grDevices 57, 58, 61
greater than 72
grep 84
grid 159, 178, 194, 222
GUI 21, 38, 39, 42

H

harmonic mean 82, 113
hatvalues 182
hclust 92, 117, 142
head 64
header 147
heading 91, 153, 156
headings 56, 146, 171
Help 19, 21, 24, 37, 38
helpfile 75, 122, 216, 217
hidden objects 59
higher-order interaction 180
high-level 11, 87, 161
hinges 111
hist 16, 42, 44, 45, 66, 124, 140, 148, 176
history 37, 154
home page 21, 36
HTML 43
hyperbolic cosine, - sine, - tangent 75

I

I() function 180, 184, 185, 186
identifier 82
identify 92, 93, 178
identifying name 82
identity matrix 77, 78
if 49, 79, 93, 94, 118, 119, 121, 124, 128, 129, 130, 133, 135, 136, 138, 145, 147, 148, 150, 151, 193, 211, 221
if-else 135, 136
ifelse 133, 135, 136, 151, 152
Im 75
importing data 55
in 124, 128, 129, 138, 157, 163, 164, 177
incorrect 71, 106
independence 139
Inf 24, 63, 70, 88
Infinity 70
influence 182, 193, 195
inherit 143
inheritance 61, 143
inherits 60, 61
initial 12, 210, 211, 212
initialize 18, 21, 27, 42
initiate 38
insert 155
installation 14, 24, 39
installing 14, 15, 37, 57
install packages 20
integer 53, 71, 77, 130, 138, 141, 142, 145
integers 40, 71, 223
integration 85
interaction 170, 180, 186, 196, 197, 198
interaction plot 186, 198
interaction.plot 170
interactive 22, 92, 149
interface 38, 39
intermediate 17, 120, 124, 156
interp 90, 178
interpolate 178
interpreter 17, 46
interpretive computer language 17
interrupt 120, 149
inverse and generalized inverse 86
inverse of the matrix 67
invisible 120, 146, 158
is.data.frame 65

is.factor 118, 119, 180
is.finite 193
is.list 65, 108
is.matrix 62, 65, 121
is.na 71, 86, 97, 98, 128, 132, 141, 147, 172
is.nan 71
is.null 118, 119, 151
is.numeric 119, 147, 151, 152
is.ordered 180
is.xxx 62, 64
isoreg 117, 142
iterative algorithm 220
iterative procedures 181, 212
iterative scheme 220

K

kable 45
keys 48, 154
knit 46
knitr 43, 45
kronecker 53, 59, 60, 61, 77
kurtosis 81

L

.Last 40, 123, 149
label 15, 16, 23, 43, 45, 91, 93, 174
labels 83, 88, 91, 92, 93, 126, 155, 166, 169, 170, 182
lapply 134, 140, 150, 151, 152
las 45
last.dump 149
latex 45
lattice 44, 159, 188
layout 43, 160, 161, 176
Layout of graphics 160
layouts 175
lazy evaluation 129
ldeaths 160
legend 94, 118, 119, 165, 166, 167, 170, 178, 188, 194
legends 165, 166
len 91, 95
length 53, 54, 55, 56, 62, 64, 65, 69, 73, 85, 90, 95, 98, 124, 128, 135, 164, 172, 173, 175, 177, 220, 221
less than 72
letters 26, 52, 82, 84, 125, 138
level 71, 83, 181
levels 53, 64, 65, 84, 88, 91, 127, 166, 170, 171
lexical scope 122
lgamma 79
libraries 120
library 24, 27, 38, 42, 45, 55, 56, 58, 130, 153, 177
likelihood 208, 212
linear model 204
linear models 187
linear regression 183
line breaks 85
line graph 112, 171
lines 50, 74, 86, 89, 93, 112, 124, 158, 163, 164, 170, 174, 187, 191, 192, 193, 200, 203, 204, 210
linewidth 45
Linpack 130
Linux 36
Lisp 11
list 16, 24, 36, 50, 55, 59, 61, 62, 65, 77, 78, 80, 82, 85, 91, 104, 105, 106, 108, 109, 118, 119, 120, 122, 124, 127, 135, 138, 139, 141, 146, 149, 152, 154, 157, 159, 160, 169, 173, 175, 220
list of arguments 16
lm 45, 73, 117, 142, 143, 182, 183, 184, 185, 186, 187, 188, 191, 193, 194, 196, 198, 201, 202, 203, 204
load 19, 35
loaded 24, 56, 59, 131
loading 57, 59, 129
Load workspace 19, 35
local object 122, 123
local variable 122
locator 92, 93, 165, 178
loess 189
log 75, 117, 150, 151, 168, 208, 213, 215, 216, 217, 219, 221
Logarithmic axes 168
logarithms 79
Logic 24
logic 55
logical and 24

logical condition 135
logical expressions 138
logical matrices 104
logical operators 24, 71, 72
logical subscripting 97, 98
logical vectors 62
logLik 182
log-likelihood 208
lognormal 66
lookup 153
loop 138, 139, 140
loops in R 138
lowess 193
low-level plotting commands 161
low-level plotting functions 163
low-level plotting instructions 87
lp 217, 219, 221
lpSolve 206, 213, 217
ls 16, 25, 55
lty 163, 164, 170
lwd 164

M
.Machine 24, 40, 71, 207, 211
Mac 36
machine 14, 15, 71, 131
mai 89
main 14, 45, 56, 87, 88, 89, 95, 96, 117, 127, 163, 164,166, 170, 176, 180, 186, 196, 197, 198
main effect 197
main effects 180, 186, 196, 198
main title 95
maketitle 43
mantissa 71
Manuals 21
mapping function 132, 134, 135
mapply 134
mar 44, 45, 89
MARGIN 144
margins 87, 89, 161
mask 121
MASS 27, 42, 56, 58, 164, 165, 183, 187, 189, 193
match 80, 121, 129, 136, 156

mathcal 44
mathematical expressions 95, 125, 126
mathematical functions 24, 75
matlines 163
matmult 129, 130, 131
matplot 163
matpoints 163
matrix 21, 54, 61, 62, 65, 67, 69, 73, 76, 77, 78, 79, 81, 86, 93, 98, 99, 100, 101, 102, 103, 104, 107, 108, 109, 110, 112, 113, 114, 116, 121, 129, 130, 132, 136, 139, 140, 143, 145, 147, 148, 163, 182, 212, 213, 217, 219
matrix argument 77, 78, 100
matrix calculations 76
matrix multiplication 129, 130
matrix object 67, 112
max 56, 66, 71, 92, 124, 156, 157, 164, 166
maximum likelihood 208, 212
maxit 187, 212
maxlen 124, 125
mdeaths 160, 162, 176
mean 15, 16, 21, 28, 42, 44, 56, 61, 66, 67, 70, 77, 81, 82, 85, 88, 91, 98, 106, 110, 112, 113, 122, 123, 132, 135, 138, 141, 148, 150, 151, 156, 169, 195, 196, 198
median 64, 169, 184, 221
medpolish 117, 142
menu 24, 28, 36, 38, 42, 56, 146, 147, 148
message 18, 32, 39, 51, 58, 62, 69, 95, 106, 119, 123, 150
Metafile 43
method 11, 118, 129, 142, 143
methods 22, 53, 57, 58, 59, 61, 95, 117, 142, 143, 166, 182
mfcol 87, 160, 176
mfg 160, 176
mfrow 87, 148, 159, 160, 169, 170, 187
mgcv 121, 180, 188
mgp 45
MiKTeX 46
min 56, 66, 157, 217, 219, 221
minimum 40, 110, 121, 122, 133, 156, 178, 190, 217, 220
Minitab 154
minus 68, 190
Misc 19
missing 63, 70, 78, 118, 119, 128, 133, 146, 178, 181, 198
missing value 63, 70
mlm 117, 142
Mod 75
mode 53, 55, 62, 63, 64, 65, 85, 105, 106, 108, 109, 124, 125, 130, 141, 142
model 180, 181, 182, 183, 184, 185, 186, 187, 189, 190, 191, 193, 195, 196, 197, 199, 200, 202, 204, 212
model.frame 182
model.matrix 182
model fitted 184
model formulae 180, 181
model notation 181
model selection 193
model specification 181
modes and attributes 63
modifier 26
modifiers for regular expressions 26
modifiers operating on characters 26
modify 98, 183
modulus 68, 74
mtext 163
multiple values 122
multiplication 129, 130, 140
multivariate normal 56, 67
multivariate normally 56
mvrnorm 56
mvtnorm 56

N

n 15, 26, 62, 85, 92, 93, 94, 119, 129, 130, 132, 145, 146, 147, 148, 151, 163, 164, 166, 167, 170, 207
NA 24, 63, 70, 71, 73, 95, 97, 98, 102, 106, 117, 128, 129, 136, 152, 189, 192
na.omit 98

name 11, 15, 16, 18, 22, 23, 24, 25, 28, 32, 33, 34, 50, 52, 53, 55, 58, 59, 61, 64, 74, 75, 78, 79, 82, 84, 93, 94, 121, 122, 127, 128, 134, 136, 138, 141, 142, 153, 155, 157, 181, 182
name clashes 121
named arguments 75
named elements 77
named objects 57
names 23, 25, 27, 50, 52, 53, 55, 59, 78, 80, 84, 86, 88, 89, 90, 95, 98, 104, 109, 110, 125, 126, 127, 156, 157, 181, 182, 187, 189
names are case sensitive 23
namespace 59, 95, 117, 118, 121, 142
naming conventions 52
naming objects 52
NaN 24, 63, 70, 71
nchar 84, 85, 156
ncol 62, 69, 73, 93, 95, 101, 102, 104, 107, 110, 121, 129, 130, 136, 139, 157, 217, 219, 220, 221
nested effects 180
nested for loops 140
nested if - else 138
nested loops 139
new 18, 20, 23, 28, 33, 41, 50, 59, 75, 87, 121, 136, 146, 148, 155, 160, 171, 177, 189
Newton-Raphson 206, 209, 210, 211, 212
next 18, 29, 49, 94, 120, 139, 160, 161, 175
next and break 139
next graph 160, 161
NextMethod 119, 143
next page 29
next tab 18
nls 117, 142, 189, 190, 191
nomatch 136
non-central 94
non-centrality 94
non-visible functions 95, 117, 142
no.readonly 160, 164, 165, 169, 176
normal distribution 15, 66, 67, 82, 85, 112, 174
normalize 132
normal quantile 191
not a number 63, 70
notation 70, 71, 116, 181
not available 59, 178
not a vector 100
notch 89, 90
not equal to 41, 202
not evaluated until 129
not exported 95, 121
not visible 49, 95, 118
nrow 54, 62, 69, 95, 99, 101, 102, 104, 107, 110, 127, 129, 130, 139, 217, 219, 220, 221
nticks 95
NULL 54, 65, 95, 117, 118, 145
number of arguments 124
number of characters 84
number of columns 139
number of decimals 156
number of elements 220, 221
number of levels 197
number of lines 89
number of rows 220, 221
numbers 40, 52, 62, 71, 75, 81, 100, 113, 139, 150, 151, 156, 158
numeric and complex vectors 24
numeric codes 54, 83
numeric concomitant variable 202
NumericConstants 24
numeric subscript 101
numeric subscripting 101
numeric variable 180

O

object 15, 16, 22, 23, 24, 25, 36, 40, 49, 52, 53, 54, 55, 58, 59, 61, 62, 63, 64, 67, 69, 71, 73, 74, 78, 79, 83, 86, 91, 95, 97, 105, 106, 112, 113, 118, 120, 121, 122, 123, 127, 128, 135, 142, 143, 144, 146, 149, 153, 158, 164, 178, 181, 182, 183, 185, 186, 194, 195, 196, 213
object .Machine 71
objective function 213, 215, 216, 217, 218, 220, 221
object matching 118
object name clashes 121
object-oriented 142, 143

objects 16, 20, 25, 27, 29, 36, 50, 51, 52, 59, 78, 84, 127, 153
objects between computers 154
objects for transport 154
objects in the workspace 49, 149
objects passed as arguments 126
objects with identical names 52
octave 153
odbcConnectExcel 153
off 87, 140, 149, 164, 165, 175
omd 176
omit 98
one-way 195, 199, 200, 201, 204
on.exit 123, 148, 149, 164, 165, 169, 176
open an R session 36
open graphic devices 87
open the R Commander 38
open the required R session 39
open the RStudio development 36
operating system 14, 158
operator 21, 24, 59, 68, 72, 73, 90, 121, 122, 127, 143, 180, 181
optim 206, 212, 213
optimization 206, 213, 220, 221
optional argument 61
options 24, 28, 31, 40, 42, 43, 87, 123, 149, 154, 156
order 24, 33, 34, 36, 43, 56, 78, 79, 95, 98, 104, 121, 126, 127, 128, 131, 134, 135, 142, 146, 178, 180, 192, 193, 196, 197, 201, 206
ordered 33, 78, 180
organization of data 61
oriented 11, 22, 142, 143
orthogonal matrix 78
orthonormal basis 114
orthonormal matrix 78
outer 77, 91, 95, 96, 133, 140, 161, 172
outer margin 161
overflow 40, 71
over-written 23

P

p-value 184, 185, 201
package 24, 27, 38, 40, 42, 43, 53, 56, 57, 58, 59, 61, 75, 90, 91, 93, 95, 121, 127, 153, 159, 164, 177, 178, 183, 187, 188, 189, 193, 212, 216, 217, 220, 221
packagename 24
pairs 91, 95, 127, 134, 148
panel 36, 37, 117, 189, 192, 193
panels 192
par 44, 45, 87, 117, 134, 148, 149, 159, 160, 164, 165,166, 169, 170, 172, 175, 176, 187, 192, 193, 212
parallelism of the two regression lines 187
parameter 87, 89, 84, 114, 122, 123, 148, 159, 160, 165, 166, 190, 191, 196, 200, 208, 211, 212
Paren 24
parent 56, 57, 58, 59, 146
parent.env 58
parent environment 57, 59
parent frame 146
parentheses 16, 24, 73, 74, 122
parse 52, 95, 96, 127, 142
partial matching 119
paste 19, 48, 84, 85, 94, 95, 96, 127, 141, 156, 157, 158, 166
pat 27
path 18, 24, 27, 28, 31, 33, 34, 37, 40, 42, 52, 53, 56, 57, 58, 59, 61, 62, 78, 93, 120, 123, 131, 153, 155, 177, 181
pattern 25, 55, 59, 141, 142, 143, 144, 178
pch 166, 177, 192, 193
perl 25
permutations 24, 81
persp 91, 95, 96, 133, 171, 172, 173, 178
perspective 95, 96, 133, 174
pf 203
phi 91, 95, 96, 172 pi 91, 95, 114, 177
pie 91
platform-independent 87
plot 87, 90, 92, 93, 94, 95, 96, 114, 116, 117, 118, 119, 120, 122, 123, 126, 127, 133, 142, 148, 164, 166, 167, 168, 170, 171, 173, 174, 176, 177, 178, 182, 183, 186, 187, 188, 189, 190, 191, 192, 196, 198, 199
plot.acf 117, 142
plot.default 117, 142
plot.dendrogram 117, 142
plot.density 117, 142
plot.ecdf 117, 142
plot.factor 117, 118, 119, 142
plot.formula 117, 142
plot.function 117, 142
plot.histogram 117, 142
plot.isoreg 117, 142
plot.lm 117, 142, 182
plot.medpolish 117, 142
plot.mlm 117, 142
plot.new 177
plot.prcomp 117, 142
plot.princomp 117, 142
plot.stepfun 117, 142
plot.table 117, 142
plot.ts 117, 142
plot.window 177
plot function 117
plotmath 174
plot region 123
plotting character 166
plotting commands 161, 163
plotting functions 163
plotting instructions 87, 88
plotting method 118
pmat 173
pmatch 80, 119
pmax 81
pmin 81
png 93, 94
poisson 187, 208
polar coordinates 114, 115
poly 193

polygon 163
polynomial 188, 192
pos 25, 27, 50, 55, 59, 61, 78, 84, 166
position 24, 25, 27, 57, 58, 75, 93, 160, 161
Powerpoint 42, 43
prcomp 117, 142
preceded by a backslash 26
preceded by double backslashes 26
precedence of operators 24
predict 143, 182, 193, 194
preferences 21, 42
pretty 84
prev 175
previous command 48
primes 148
princomp 117, 142
print 122, 149, 156, 175, 182, 183
prmatrix 156
probability 66, 82, 94, 124, 171
proc.time 140
profile 117, 142
proj 182
project 11, 14, 33, 34, 36, 37, 39, 49, 79, 114
prompt 15, 16, 19, 23, 24, 29, 31, 40, 48, 50, 75, 123, 142
pty 166

Q

q 16, 23, 41, 82, 119, 129, 130, 212
qqline 163, 191
qqnorm 92, 163, 182
qqplot 92
qr 77, 78, 143, 182
qt 28
quadprog 213, 216
quantile 82, 111, 169, 191
quantile plot 191
quotation marks 24
quote 24
quotes 21, 50, 93

R

.RData 16, 22, 33, 34, 36, 37, 39, 48, 49, 50, 55, 149, 154
radians 114
radius 93, 114
random permutations 81
random sample 15, 67, 81, 82, 112, 174, 212
random seed 132
random starts 220, 221
range 26, 81, 95, 153, 170
rank 67, 79, 113
rapply 134
rbind 76, 78, 110, 133, 217, 219
rbinom 66
rcauchy 138
R CMD 130
Rcmdr 38
R command 177, 196
R Commander 38, 39
R console 19, 20, 28, 29, 30, 31, 32, 36, 154
R data 28, 111, 160
R database 25
R data set 28
Re 75
read.arff 153
read.csv 55, 153
read.dbf 153
read.dta 153
read.epiinfo 153
read.mtp 153
read.octave 153
read.S 153, 223
read.spss 153
read.ssd 153
read.systat 153
read.table 40, 55, 56, 62, 76, 147, 153
read.xport 153, 223
readline 41, 93, 94

read-only 146
readonly 160, 164, 165, 169, 176
real numbers 40, 71
Recall 145, 177
recall previous commands 154
record 86, 149, 154
records 86
recover 32, 143, 149
recursion 145
recycling principle 62, 69, 80, 111
re-editing 154
region 110, 123, 158
regression analysis 183
regression coefficients 45, 200, 204
regression equations 202, 204
regression lines 187, 200, 203, 204
regression model 189, 190
regression parameters 200
regression relationship 200, 201
regression terms 180
regular expression 25, 26
remove 50, 55, 58, 78
removed 16, 57
rep 56, 90, 101, 139, 166, 216, 217, 219, 220, 221
repeat loops 139
replace 24, 80, 127, 128, 156, 178
replacement function 127, 128
report 42, 151, 152
req 147, 148
require 44, 93, 161, 164, 165
reserved word 21
reset 123, 160
residuals 182, 189, 191, 193, 194, 196, 199
response pattern 143, 144
restarts 220
restore 153
return 18, 55, 58, 67, 77, 113, 119, 120, 121, 126, 128, 136, 148, 173, 193
returns a density value 82
rev 56, 140
R evaluator 17, 18, 23, 32, 58, 121
R expressions 135, 177

R function 16, 21, 25, 28, 29, 31, 41, 67, 78, 86, 110, 113, 114, 122, 126, 129, 130, 139, 144, 145, 146, 148, 154, 158, 174, 177, 190, 194, 207, 209, 211
rgl 93, 94, 177
rgl.set 94
R graphics 87, 114, 159
R graphs 161
Rgui 33
Rhistory 33, 34, 154
R instructions 18, 24, 25, 42, 67, 68, 74, 86, 112, 154, 219
R language 11
rlnorm 66
rm 16, 36, 50, 52, 55, 95
R modelling 179, 181
rmvnorm 56
R names 23
rnorm 15, 16, 44, 45, 50, 56, 61, 85, 93, 120, 122, 124,140, 150, 151, 163
rnw 43, 46
R object 15, 16, 64, 86, 113, 121, 144, 153, 158, 195, 196, 213
RODBC 153
round 42, 44, 76, 87, 111, 156, 172
rounding and truncating 76
routines 129
R output 42, 43
row 56, 76, 91, 100, 101, 104, 110, 121, 132, 143, 160, 161
rownames 62, 76, 92, 93, 98, 110, 127, 157
row-wise 76, 160
R package 38, 43, 56, 57, 153, 177
rpart 180, 189
R programming 69, 87, 97, 132, 145
R-project.org 14
R projects 33, 78
R prompt 15, 16, 19, 23, 24, 29, 31, 40, 48, 142
R script 28
R session 15, 16, 18, 21, 23, 27, 36, 38, 39, 40, 42, 48, 49, 57, 154, 158
Rsolnp 206, 213, 220, 221
R-squared 184, 185
rstandard.lm 182
R strings 142
rstudent.lm 182
RStudio 36, 38, 39
R text 28, 42
rug 163
run 32 bit or 64 bit applications 15
run a SAS program 223
runif 56, 66
run the function 29
run the script 28, 29
R workspace 36, 39, 46

S

S 11, 12, 14, 52, 129, 153, 154, 222, 223
S3 11, 118, 143
S4 11, 22, 143
sample 15, 28, 66, 67, 81, 82, 86, 108, 112, 113, 122, 146, 150, 151, 163, 174, 178, 195, 212
sapply 134, 140
SAS 154, 223
save 14, 16, 23, 41, 49, 50, 78, 79, 149, 154
save.image 49, 50
Save workspace 19, 35
scale 77, 92, 95, 124, 132, 143, 165, 168, 194, 211, 212
scan 55, 76, 79, 85, 153
scatterplot 92, 93, 112, 174, 178, 190, 199, 201, 203, 204
scientific notation 70
scope 122, 185, 186
score statistic 211
screen 55, 116, 155, 161
script 19, 28, 29, 30, 37, 120, 122, 154
script window 28, 29, 154
sd 15, 28, 56
search 21, 24, 25, 27, 37, 40, 42, 52, 53, 57, 58, 59, 61, 78, 82, 84, 120, 121, 123, 146, 175, 177, 181, 206
search list 24, 59, 61, 146
search path 24, 27, 37, 40, 42, 52, 53, 57, 58, 59, 61, 78, 120, 123, 177, 181
secondary default prompt 50
secondary R prompt 15

second position on the search path 24
seed 44, 67, 82, 112, 132
semi-colon 18, 50
sep 94, 96, 141, 147, 155, 166, 202
separate project 34
separate workspaces for each project 49
seq 50, 56, 91, 92, 93, 95, 114, 121, 172, 177
setClass 142
set.seed 44, 67, 82
setwd 49, 50
shade 95, 163, 172
show.output.on.console 158
sign 23, 50, 75, 153, 207
signif 156
silent 151
simple 95
simulate 182
sin 91, 114, 172, 177
single backslash 18
single quotes 50
single square brackets 104
singular value decomposition 78
sinh 75
sink 154
size 43, 66, 67, 77, 82, 92, 93, 102, 112, 114, 115, 116, 140, 145, 174, 212
skeleton 75
skewness 81
sleep 18, 28, 29, 177
slots 143
smooth.spline 164, 192
snapshot3d 94
solnp 221
solve 67, 78, 208, 216
sort 79, 107, 140, 163
sorting methods 166
source 11, 113, 154
span 193
spar 164
spec 117, 142
special argument ... together 148
special operator ... 24
special symbols 174
spheres3d 93
spineplot 119
spline 164, 188, 192
split 85, 88, 89, 90, 92, 134, 161
S-Plus 11, 12, 13, 52, 222, 223
spss 153
sqlFetch 153
sqlTables 153
sqrt 28, 75, 95, 172
square brackets 26, 104, 106
square matrix 77, 78
square plot region 123
ssd 153, 154, 223
standard deviation 15, 77, 85, 196
standard error 67, 184, 185
start 14, 15, 21, 49, 50, 57, 114, 117, 191, 221
starting an escape sequence 24
starting value 190
start values 191, 221
Stata 154
state.x77 data set 72, 81, 82, 83, 88, 92, 110, 134, 156, 158, 178, 192
statistical analyses 11, 38, 42
statistical distributions 82
statistical functions 81
statistical hypothesis 196
statistical model 180
statistical model in R 180
statistical modelling 109, 179, 180, 182
statistical modelling with R 179
stats 57, 58, 61, 84, 111, 193, 212
stdres 193, 194
step 34, 193, 194, 206
stepfun 117, 142
stop 18, 52, 119, 129, 130, 145, 147, 150, 151, 207, 220, 221
storage.mode 130, 141, 142
str 64, 74, 121, 196, 217
stringsAsFactors 108
strsplit 85
studres 193, 194

sub 95, 117, 170
subroutine 130
subscripting lists 106
subscripting matrix 101, 102, 103
subscripting methods 99
subscripting numeric matrix 101
subscripting operations 98
subscripting tools 139
subscripting vectors 97
subset 104, 181, 184, 196
substitute 95, 125, 126, 127, 146
substring 84, 156, 166
sum 62, 66, 67, 73, 85, 129, 139, 140, 203, 215, 216, 217, 220, 221
summary 45, 64, 111, 149, 181, 182, 183, 184, 187, 189, 196
surface3d 177
survival time 212
svd 78, 86, 113, 114
sweep 132, 133, 140, 219
switch 16, 71, 135, 138
symbol # 23, 50, 51, 122, 127
symbols 23, 106, 122, 174
syntax error 29, 31, 32
syntax mistake 31
Sys.getenv 158
sys.parent 146
Sys.sleep 18, 177
systat 153
system 11, 12, 14, 15, 21, 22, 38, 140, 158, 163
system.time 140, 163

T

t 18, 28, 52, 77, 110, 140, 158, 177, 216
t.test 110
tab 18
table 40, 55, 56, 62, 76, 84, 117, 119, 136, 141, 142, 147, 153, 156, 157, 197
tan, tanh 75
tangent 75, 210
tapply 88, 91, 135, 140, 169, 196
target string 26

tcl 45
terminated 16, 59, 123
terminating 23, 158
terms, main effects, interaction effects 180
test 110, 139, 187, 194, 197, 198
testable 196
text 18, 28, 29, 31, 40, 42, 43, 45, 49, 52, 55, 56, 62, 86, 92, 93, 95, 118, 119, 125, 126, 142, 146, 147, 154, 158, 166, 170, 173, 174, 189
text editor 28, 31, 40, 43, 154
text message 18
texttt 45
theta 91, 92, 95, 96, 114, 172
three-dimensional graph 91, 171
ticktype 91, 95, 96, 172
tilde 73, 90, 180, 202
time 40, 61, 92, 111, 116, 121, 131, 140, 149, 158, 163, 166, 190, 196, 212
time series 92
Tinn-R 28
title 32, 43, 88, 95, 134, 147, 148, 170, 174, 176
trace 149, 189
traceback 149
tracing 149
traditional R graphics 87, 159
trans3d 173
transpose 77
transport 154
tree-based model 180
trellis graphics 159
triangular matrix 77, 78
trim 138, 141
trimmed mean 81, 132 triple-colon 59
TRUE 25, 44, 52, 55, 56, 59, 60, 61, 62, 65, 71, 73, 77, 87, 89, 90, 95, 97, 99, 102, 104, 108, 109, 117, 119, 120, 124, 134, 135, 139, 151, 154, 155, 157, 158, 160, 164, 165, 169, 176, 189
trunc 76
truncating 76
try 150, 151, 152
try-error 151
ts 62, 92, 117, 142, 176
tskernel 117, 142
TukeyHSD 117, 142
two-index 99
two-way 83, 139, 186, 187, 195, 196, 199
txt 18, 146, 147, 153, 158
type 22, 40, 91, 93, 114, 117, 163, 164, 166, 170

U

unary 24, 68, 72, 74
unclass 63, 83, 84
undebug 149
undefined 70, 128, 129
underflow 40, 71
underscore, underscores 52
unevaluated 126, 127, 142
unexpected 32, 50, 51
unfinished 15
uniform distribution 66
unique 80, 143, 144, 206
uniroot 209
Unix 11, 25, 223
unlist 85, 90, 109, 152
unquoted 95
unspecified number of arguments 124
untrace 149
up and down arrow 48
update 183, 184
updating 37
upper and lower case 26, 52
upper triangular 77,78
usage of .First 41
usage of grep 84
usage of identify 92
usage of invisible 120
usage of locator 93
usage of match 80, 136
usage of the double colon 59
usage of the function interp 90
usage of the function layout 161
usage of the function optim 206
usage of the function matrix 110
usage of the function stop 119

usage of the function ts 62
usage of the function warning 119
usage of the function warnings 119
usages 78, 121
use 21, 32, 43, 50, 52, 56, 71, 73, 78, 83, 84, 88, 89, 91, 92, 93, 95, 110, 111, 113, 116, 119, 121, 122, 124, 125, 130, 131, 132, 133, 134, 135, 140, 142, 145, 155, 158, 160, 164, 166, 174, 175, 177, 178, 180, 182, 183, 187, 188, 189, 191, 192, 196, 209, 212, 213, 217
use of 24, 26, 28, 40, 50, 52, 55, 59, 66, 68, 69, 71, 73, 74, 75, 78, 79, 80, 87, 95, 98, 104, 107, 114, 124, 129, 148, 149, 154, 156, 166, 183, 188, 193
UseMethod 117, 142
usepackage 43
user 15, 16, 24, 39, 59, 78, 93, 95, 116, 140, 146, 148, 168, 194
using any text editor 28
using a script 28
using as start values 221
using fix 31, 129
using floating point representation 71
using if - else statements 138
using named arguments 75
using proc.time 140
using R 68, 146
using script files 28
using scripts or fix 120
using subscripting 97
using the command str 121
using the function traceback 149
using the subscripting tools 139
using vectors as subscripts 101
usr 165, 170
utils 57, 58, 61

V

v, V 18, 50, 51, 78, 113, 114, 133
value 70, 74, 82, 85, 98, 114, 116, 119, 120, 121, 123, 126, 128, 129, 136, 182, 190, 192, 203
values 24, 25, 26, 56, 63, 70, 74, 77, 78, 79, 80, 95, 98, 101, 102, 119, 122, 123, 124, 128, 129, 138, 155, 159, 160, 181, 190, 193, 196, 198, 199, 211, 212, 220, 221
var 28, 84, 140, 148
variable number of arguments 124
variables 55, 64, 66, 67, 81, 83, 84, 85, 86, 92, 110, 112, 116, 122, 134, 135, 146, 147, 149, 156, 158, 179, 180, 181, 186, 192, 194, 195, 196, 198, 199, 217
variance 122, 183, 195, 198, 201, 202
vcov 182
vector 15, 50, 62, 67, 68, 75, 77, 78, 80, 82, 85, 88, 89, 98, 99, 100, 101, 103, 109, 112, 113, 121, 122, 133, 135, 136, 138, 145, 148, 155, 211, 212, 217
vector arguments 68
vector calculations 62
Vectorize 134
vectorized 69, 111, 132
vectorizing 69
vectors 24, 55, 62, 63, 69, 77, 97, 100, 101, 109, 114, 126, 127, 134, 138
vertical 18, 114
visibility 95, 118
volcano 96

W

warn 149
warning 23, 55, 69, 119, 149, 211
warnings 119
Weibull-distribution 211
weights 181, 189, 215
what 85
where 60, 61, 121
which 121
while 93, 139, 207, 211
while loops 139
whiteside 187
width 45
window 16, 19, 28, 29, 30, 31, 32, 33, 43, 84, 87, 89, 93, 112, 115, 116, 123, 154, 161, 166, 177
windows 14, 16, 28, 29, 33, 37, 87, 175
with 61, 109, 183, 196, 197, 198

within-groups 195
without warning 23, 55, 69
working directory 16, 22, 23, 33, 36, 37, 41, 49, 55, 73, 74, 123, 131
worksheets 154
workspace 16, 19, 33, 35, 36, 37, 39, 41, 46, 49, 50, 55, 57, 68, 79, 149
write 14, 15, 18, 67, 71, 79, 86, 111, 113, 114, 116, 117, 124, 126, 128, 145, 146, 153, 154, 156, 158, 174, 178, 209
writing functions in R 28, 31, 40, 111, 117, 143, 146
writing R objects for transport 154
writing a panel function 192

X

x-axis 133, 168, 178, 194
xaxt 114, 119, 166, 170
Xbase 154
xlab 88, 92, 95, 114, 117, 127, 163, 164, 166, 170, 172, 176
xlim 95, 117, 124
xpd 189, 198
xport 153, 154, 223
xtick 170
xy 168

Y

y-axis 134, 168, 194
yaxt 114, 166
ylab 88, 92, 95, 114, 117, 127, 163, 164, 166, 170, 172, 176
ylim 95, 117, 164, 166

Z

zlab 95, 172
zlim 95
zoom 177