

Pierre Lafaye de Micheaux
Rémy Drouilhet
Benoit Liquet

The R Software

Fundamentals of Programming
and Statistical Analysis

Statistics and Computing

Series Editors:

J. Chambers

D. Hand

W. Härdle

For further volumes:

<http://www.springer.com/series/3022>

Pierre Lafaye de Micheaux • Rémy Drouilhet
Benoit Liquet

The R Software

Fundamentals of Programming
and Statistical Analysis



Springer

Pierre Lafaye de Micheaux
Department of Mathematics and Statistics
Université de Montréal
Montréal, QC, Canada

Rémy Drouilhet
B.S.H.M
Grenoble, France

Benoit Liquet
School of Mathematics and Physics
The University of Queensland
Brisbane, Australia

Series Editors:

J. Chambers
Department of Statistics
Sequoia Hall
390 Serra Mall
Stanford University
Stanford, CA 94305-4065

D. Hand
Department of Mathematics
Imperial College London
South Kensington Campus
London SW7 2AZ
United Kingdom

W. Härdle
C.A.S.E. Centre for Applied
Statistics and Economics
School of Business and
Economics
Humboldt-Universität zu
Berlin
Unter den Linden 6
10099 Berlin
Germany

ISSN 1431-8784
ISBN 978-1-4614-9019-7
DOI 10.1007/978-1-4614-9020-3
Springer New York Heidelberg Dordrecht London

ISSN 2197-1706 (electronic)
ISBN 978-1-4614-9020-3 (eBook)

Library of Congress Control Number: 2013953875

© Springer Science+Business Media New York 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To the open-source community

*To all those who have contributed, are contributing and will
contribute
to the awakening of our consciousness*

*To our colleagues from Montpellier, Grenoble, Bordeaux and
ISPED*

Foreword

This book has been translated from French by Robin Ryder, who is assistant professor in the CEREMADE (Centre De Recherche en Mathématiques de la Décision) at Université Paris Dauphine (France). We are very pleased that he has agreed to make this translation.

This book is based on notes from a series of lectures given for a few years at the Institut Universitaire de Technologie Grenoble 2 in the Department of Statistics and Business Intelligence (STID, *Statistique et informatique décisionnelle*). It has therefore been “digested” first, in a very imperfect form, by the students of this department, whom we thank here. Had they not shown so much interest, this book would probably never have existed. We also thank our colleague and friend Michel Lejeune, who managed to talk to us about writing a manuscript and submitting it to Springer. It is worth pointing out the role of chance, which made the paths of the three authors cross in the same place for a few years. The human and scientific experience of this encounter was very enriching, and each author provided complementary skills which made it possible to overcome the tremendous amount of work necessary for this book. Finally, we wish to warmly thank our colleague and friend Matthieu Dubois, who is a researcher in experimental psychology and addicted to R and to Macintosh and who was the first to read the French version of the book in its almost finalized version and gave us many ideas for improvement.

The contents of this book were chosen and organized in the best possible way for them to be not only **exhaustive** but also **easy to assimilate** by the reader. This book can be used as support material for lectures on R at any level from beginner to advanced. We have paid particular attention to the form of the book, which we think should aid understanding. It can also be used as a support for self-teaching. Note that most of this book can be useful to users of any operating system. However, a few chapters are mostly meant for users of Microsoft Windows. We have also felt it useful to give, occasionally, complements aimed at users of Linux or Macintosh.

All chapters follow the same structure. A chapter begins with a small insert listing the prerequisites necessary for the chapter and a short description of the contents. All theoretical notions are explained with numerous examples and include breaks so that the reader can put into practice on a computer the recently introduced notions. Each chapter ends with an assessment section: memorandum of most important terms, followed by a section of theoretical exercises (to be done on paper), which can be used as questions for a test. A practical sheet is also given at the end of each chapter. It can be used to check that the practical aspects of the chapter have been taken in. Note that all exercises and practicals only require the contents of the previous chapters.

The structure of the book is sequential. After a short introduction (see the first part), aimed at getting the reader interested, and a description of a few data sets which will be used throughout the book to illustrate how to use R, the second part of the book is dedicated to the fundamental concepts of R: data organization, import and export, various manipulations, documentation, plots, programming and maintenance. This part should help you “learn the ropes” of R.

The third part of the book is dedicated to using R in a few mathematical and statistical settings. You should read the second part before moving on to this part, although it can be understood by users who already have a few notions in R. It covers R instructions for some of the main statistics and mathematics courses up to third-year undergraduate (e.g., it covers the baccalaureate in statistics and actuarial sciences curriculum at Université de Montréal, as well as the French IUT curriculum in statistics and business intelligence): matrix operations, integration, optimization, descriptive statistics, simulations, confidence intervals and hypothesis testing, simple and multiple linear regression and analysis of variance.

Finally, note that each statistical chapter in the third part relies on one or several real data sets, kindly made available by ISPED (*Institut de santé publique, d'épidémiologie et de développement* in Bordeaux) and described at the beginning of the book. These make learning more concrete and more attractive. We take this opportunity to thank all the teaching staff from the Public Health School of ISPED. These data, as well as several functions developed specially for this book and which are described or used here, are available in an R package associated with this book, called **TheRSoftware**. We also thank Mohamed El Methni and Taghi Barumandzadeh for the material they gave us for the chapter on ANOVA and Hubert Raymondaud for many comments he has made on our French version which allowed us to significantly improve several sections of this book.

Montréal, Canada
Grenoble, France
Brisbane, Australia

Pierre Lafaye de Micheaux
Rémy Drouilhet
Benoit Liquet

Alternative Order of Reading

We have used the symbol † to make explicit more difficult or less fundamental sections, which you can skip at first read without prejudicing your understanding or mastering of R.

Note that this book was conceived for students from a mathematical or statistical background. However, students or scientists from a more “applied” background can also use it: for these readers, we propose a different order of reading, as follows; the difficult sections should also be omitted.

Part A: The Basics of R

- (a) Basic concepts; data organization (Chap. 3)
- (b) Import-export and data production (Chap. 4)
- (c) Data manipulation (Chap. 5)
- (d) R and its documentation (Chap. 6)
- (e) Techniques for plots (Chap. 7)
- (f) Maintaining sessions (Chap. 9)

Part B: Elementary Statistics

- (a) Random variables, distributions and simulation (Chap. 12)
- (b) Descriptive statistics (Chap. 11)
- (c) Confidence intervals and hypothesis testing (Chap. 13)
- (d) Simple and multiple linear regression (Chap. 14)
- (e) Elementary analysis of variance (Chap. 15)

Part C: Advanced Concepts

- (a) Basic mathematics: matrix operations, integration, optimization (Chap. 10)
- (b) Programming in R (Chap. 8)

Inserts

We have tried to be careful with the presentation of the book (the form), to make the information (the content) more easy going. Inserts are located at strategic points in the book, to help bring out some of the important information and make notions easier to understand. These inserts are distinguished by icons in the margin.

Tip

Additional information about the topic under study.



Warning

Important point you should pay attention to.



Note

Advice and practical tricks.



See also

Refers to another chapter or a website.



 Advanced users

Advanced elements. You can omit these at first.

 Linux

Information for Linux users.

 Mac

Information for Macintosh users.

Solutions to Exercises and Practicals

Solutions to exercises and practicals are given on the book's website (<http://www.biostatisticien.eu/springeR>).

Other projects, more ambitious than the practicals, will be made available on the same website.

Font Conventions

- The letter **R** refers to the R software.
- We use *italics* for words borrowed from Latin or French or for abbreviations.
- We use a **fixed width font** (`Verbatim` environment) for R instructions.
- We use **SMALL CAPS** for data sets and **sans serif** for the name of the file including these data sets. This font will be used for all file names and folder names used in this book.

Contents

Foreword	vii
List of Figures	xxvii
List of Tables	xxxii
Mathematical Notations	xxxiii
Part I Preliminaries	
1 Introducing R	3
1.1 Presentation of the Software	3
1.1.1 Origins	3
1.1.2 Why Use R?	3
1.2 R and Statistics	5
1.3 R and Plots	5
1.4 The R Graphical User Interface	7
1.5 First Steps in R	7
1.5.1 Using RCommander	7
1.5.1.1 Launching RCommander	8
1.5.1.2 Handling Data with RCommander	8
1.5.1.3 A Few Statistical Tasks with RCommander	13
1.5.1.4 Adding Functionalities to the RCommander Interface	18
1.5.2 Using R with the Console	19
1.5.2.1 The Strength of R Shown on an Example	19
1.5.2.2 A Brief Introduction of R Syntax Through Some Instructions to Type	23
2 A Few Data Sets and Research Questions	29
2.1 Body Mass Index of Children	29
2.2 Weight at Birth	30

2.3	Intima–Media Thickness	31
2.4	Diet of Elderly People.....	32
2.5	Study Case of Myocardial Infarction	33
2.6	Summary Table of Use of Data Sets	33

Part II The Bases of R

3	Basic Concepts and Data Organisation	37
3.1	Your First Session	37
3.1.1	R Is a Calculator	38
3.1.2	Displaying Results and Variable Redirecting	39
3.1.3	Work Strategy	40
3.1.4	Using Functions	43
3.2	Data in R	46
3.2.1	Data Nature (or Type, or Mode)	46
3.2.1.1	Numeric Type (<code>numeric</code>)	46
3.2.1.2	† Complex Type (<code>complex</code>)	47
3.2.1.3	Boolean or Logical Type (<code>logical</code>)	48
3.2.1.4	Missing Data (<code>NA</code>)	48
3.2.1.5	Character String Type (<code>character</code>).....	49
3.2.1.6	† Raw Data (<code>raw</code>)	50
	Summary	50
3.2.2	Data Structures	50
3.2.2.1	Vectors (<code>vector</code>)	51
3.2.2.2	Matrices (<code>matrix</code>) and Arrays (<code>array</code>)	52
3.2.2.3	Lists (<code>list</code>)	53
3.2.2.4	The Individual×Variable Table (<code>data.frame</code>)	55
3.2.2.5	Factors (<code>factor</code>), Ordinal Variables (<code>ordered</code>)	56
3.2.2.6	Dates	57
3.2.2.7	Time Series	57
	Summary	58
	Memorandum	59
	Exercises	59
	Worksheet	60
4	Importing, Exporting and Producing Data	63
4.1	Importing Data	63
4.1.1	Importing Data from an ASCII Text File	63
4.1.1.1	Reading Data with <code>read.table()</code>	64
4.1.1.2	Reading Data with <code>read.ftable()</code>	67
4.1.1.3	Reading Data with the Function <code>scan()</code>	68

4.1.2	Importing Data from Excel or the Open Office Spreadsheet	69
4.1.2.1	Copy-Pasting	69
4.1.2.2	Using an Intermediary ASCII File	70
4.1.2.3	Using Specialized Packages	70
4.1.3	Importing Data from SPSS, Minitab, SAS or Matlab	70
4.1.4	Large Data Files	71
4.2	Exporting Data	72
4.2.1	Exporting Data to an ASCII Text File	72
4.2.2	Exporting Data to Excel or OpenOffice Calc	72
4.3	Creating Data	73
4.3.1	Entering Toy Data	73
4.3.2	Generating Pseudo-Random Numbers	74
4.3.3	Entering Data from a Hard Copy	74
4.4	† Reading/Writing in Databases	76
4.4.1	Creating a Database and a Table	76
4.4.2	Creating a Data Source Compatible with MySQL	76
4.4.3	Writing in a Table	78
4.4.4	Reading a Table	79
	Memorandum	80
	Exercises	80
	Worksheet	81
5	Data Manipulation, Functions	85
5.1	Operations on Vectors, Matrices and Lists	85
5.1.1	Vector Arithmetic	85
5.1.2	Recycling	86
5.1.3	Basic Functions	87
5.1.4	Operations on Matrices and Data.Frames	88
5.1.4.1	Information on Architecture	88
5.1.4.2	Merging Tables	89
5.1.4.3	The Function <code>apply()</code>	93
5.1.4.4	The Function <code>sweep()</code>	94
5.1.4.5	The Function <code>stack()</code>	94
5.1.4.6	The Function <code>aggregate()</code>	95
5.1.4.7	The Function <code>transform()</code>	95
5.1.5	Operations on Lists	96
5.2	Logical and Relational Operations	97
5.3	Operations on Sets	98
5.4	Extracting and Inserting Elements	99
5.4.1	Extracting from/Inserting into Vectors	100
5.4.2	Extracting from/Inserting into Matrices	102
5.4.3	Extracting from/Inserting into Arrays	106
5.4.4	Extracting from/Inserting into Lists	106
5.5	Manipulating Character Strings	108

5.6	Manipulating Dates and Time Units	111
5.6.1	Displaying the Current Date	111
5.6.2	Extracting Dates	112
5.6.3	Operations on Dates	113
5.7	Control Flow	115
5.7.1	Conditional Instructions	116
5.7.2	Loop Instructions	118
5.8	Creating Functions	120
5.9	† Fixed-Point and Floating Point Number Representation	127
5.9.1	Representing a Number in a Base	127
5.9.2	Floating Point Representations	128
5.9.2.1	Definitions	128
5.9.2.2	Limitations of This Representation due to the Significand	129
5.9.2.3	Avoiding Some Numerical Pitfalls	130
5.9.2.4	Limitations of This Representation due to the Exponent	132
	Memorandum	134
	Exercises	134
	Worksheet	136
6	R and Its Documentation	141
6.1	Integrated Help	141
6.1.1	The Command <code>help()</code>	141
6.1.2	Some Complementary Commands	143
6.2	† Help on the Web	145
6.2.1	Search Engines	145
6.2.2	Message Boards	146
6.2.3	Mailing Lists	146
6.2.4	Internet Relay Chat (IRC)	147
6.2.5	Wiki	147
6.3	† Literature About R	147
6.3.1	Online	147
6.3.2	Printed Material	148
	Memorandum	149
	Exercises	149
	Worksheet	149
7	Drawing Curves and Plots	151
7.1	Graphics Windows	151
7.1.1	Basic Graphics Windows, Manipulation and Saving	151
7.1.2	Splitting the Graphics Window: <code>layout()</code>	153
7.2	Low-Level Drawing Functions	156
7.2.1	The Functions <code>plot()</code> and <code>points()</code>	156
7.2.2	The Functions <code>segments()</code> , <code>lines()</code> and <code>abline()</code>	158

7.2.3	The Function <code>arrows()</code>	160
7.2.4	The Function <code>polygon()</code>	161
7.2.5	The Function <code>curve()</code>	162
7.2.6	The Function <code>box()</code>	162
7.3	Managing Colours	163
7.3.1	The Function <code>colors()</code>	163
7.3.2	Hexadecimal Colour Coding	164
7.3.3	The Function <code>image()</code>	166
7.4	Adding Text	169
7.4.1	The Function <code>text()</code>	169
7.4.2	The Function <code>mttext()</code>	170
7.5	Titles, Axes and Captions	171
7.5.1	The Function <code>title()</code>	171
7.5.2	The Function <code>axis()</code>	172
7.5.3	The Function <code>legend()</code>	173
7.6	Interacting with the Plot	175
7.6.1	The Function <code>locator()</code>	175
7.6.2	The Function <code>identify()</code>	175
7.7	† Fine-Tuning Graphical Parameters: <code>par()</code>	176
7.8	† Advanced Plots: <code>rgl</code> , <code>lattice</code> and <code>ggplot2</code>	187
	Memorandum	188
	Exercises	188
	Worksheet	189
8	Programming in R	193
8.1	Preamble	193
8.2	Developing Functions	194
8.2.1	Quick Start: Declaring, Creating and Calling Functions	194
8.2.2	Basic Concepts on Functions	195
8.2.2.1	Body of a Function	195
8.2.2.2	List of Formal and Effective Arguments	195
8.2.2.3	Object Returned by a Function	198
8.2.2.4	Variable Scope in the Body of a Function	200
8.2.3	Application to the Practical Problem	202
8.2.4	Operators	202
8.2.5	R Seen as a Functional Language	204
8.3	† Object-Oriented Programming	204
8.3.1	How the Internal Object-Oriented Mechanism Works	205
8.3.1.1	Class of an Object and Declaring an Object	205
8.3.1.2	Declaring Objects and Using Methods	206
8.3.2	Back to the Practical Problem	209
8.3.3	Information About Methods	211
8.3.4	Inheriting Classes	213

8.4	† Going Further in R Programming	216
8.4.1	R Attributes	216
8.4.1.1	Attribute <code>class</code>	218
8.4.1.2	Attribute <code>dim</code>	218
8.4.1.3	Attributes <code>names</code> and <code>dimnames</code>	221
8.4.2	Other R Objects	224
8.4.2.1	R Expressions	224
8.4.2.2	R Formulae	226
8.4.2.3	The R Environment	228
8.5	† Interfacing R and C/C++ or Fortran	230
8.5.1	Creating and Running a C/C++ or Fortran Function	231
8.5.2	Calling C/C++ (or Fortran) from R	237
8.5.3	Calling External C/C++ or Fortran Libraries	242
8.5.3.1	The R API	243
8.5.3.2	The <code>newmat</code> Library	246
8.5.3.3	The BLAS and LAPACK Packages	248
8.5.3.4	Mixing C/C++ and Fortran Packages	250
8.5.4	Calling R Code from a C/C++ Program Called by R	252
8.5.5	Calling R Code from Fortran	255
8.5.6	Some Useful Functions	255
8.6	† Debugging Functions	255
8.6.1	Debugging Functions in Pure R	255
8.6.2	Error in R Code	257
8.6.3	Error in the C/C++ or Fortran Code	258
8.6.4	Debugging with GDB	259
8.6.4.1	Debugging with Emacs	262
8.6.4.2	Debugging with DDD	264
8.6.4.3	Debugging with Insight	265
8.6.4.4	Detecting Memory Leaks	270
8.7	Parallel Computing and Computation on Graphical Cards	273
8.7.1	Parallel Computing	273
8.7.2	Computation on Graphical Cards	274
	Memorandum	276
	Exercises	276
	Worksheet	278
9	Managing Sessions	283
9.1	R Commands, Objects and Storage	283
9.2	Workspace: <code>.RData</code> Files	285
9.3	Command History: <code>.Rhistory</code> Files	287
9.4	Saving Plots	288
9.5	Managing Packages	290
9.6	Managing Access Paths to R Objects	290
9.7	† Other Useful Commands	292

9.8	† Problems in Memory Management	293
9.8.1	Organization of RAM	293
9.8.2	Accessing the Memory	294
9.8.2.1	Problems Caused by Memory Management of Integers	295
9.8.2.2	Successive Allocation of Memory	296
9.8.3	Object Size in R	298
9.8.4	Total Memory used by R	299
9.8.5	A Few Recommendations	301
9.9	† Using R in BATCH Mode	302
9.10	† Creating a Simple R Package	303
	Memorandum	306
	Exercises	306
	Worksheet	307

Part III Elementary Mathematics and Statistics

10	Basic Mathematics: Matrix Operations, Integration, Optimization	313
10.1	Basic Mathematical Functions	313
10.2	Matrix Operations	315
10.2.1	Basic Matrix Operations	316
10.2.2	Outer Product	318
10.2.3	Kronecker Product	319
10.2.4	Triangular Matrices	319
10.2.5	Operators vec and Half vec	320
10.2.6	Determinant, Trace and Condition Number	320
10.2.7	Scaling and Centring Data	321
10.2.8	Eigenvalues and Eigenvectors	321
10.2.9	Square Root of a Hermitian Positive-Definite Matrix	322
10.2.10	Singular Value Decomposition	323
10.2.11	Cholesky Decomposition	323
10.2.12	QR Decomposition	324
10.3	Numerical Integration	325
10.4	Differentiation	326
10.4.1	Symbolic Differentiation	326
10.4.2	Numerical Differentiation	327
10.5	Optimization	327
10.5.1	Optimization Functions	327
10.5.2	Roots of a Function	331
	Memorandum	333
	Exercises	333
	Worksheet	334

11 Descriptive Statistics	339
11.1 Introduction	339
11.2 Structuring Variables According to Type	340
11.2.1 Structuring Qualitative Variables	341
11.2.2 Structuring Ordinal Variables	342
11.2.3 Structuring Discrete Quantitative Data	342
11.2.4 Structuring Continuous Quantitative Variables	343
11.3 Data Tables	343
11.3.1 Individual Data Tables	343
11.3.2 Tables of Counts and Frequency Tables	343
11.3.3 Tables of Grouped Data	344
11.3.4 Cross Tabulation	344
11.3.4.1 Contingency Tables	344
11.3.4.2 Joint Distribution	345
11.3.4.3 Marginal Distributions	346
11.3.4.4 Conditional Distributions	346
11.4 Numerical Summaries	347
11.4.1 Summaries of the Location of a Distribution	348
11.4.1.1 Modes	348
11.4.1.2 Median	348
11.4.1.3 Mean	350
11.4.1.4 Quantiles	350
11.4.2 Summaries of the Dispersion of a Distribution	350
11.4.3 Summaries of the Shape of a Distribution	351
11.5 Measures of Association	352
11.5.1 Measures of Association Between Two Qualitative Variables	352
11.5.1.1 Pearson's χ^2 Statistic	352
11.5.1.2 ϕ^2 , Cramér's V and Pearson's Contingency Coefficient	353
11.5.2 Measures of Association Between Ordinal Variables (or Ranks)	354
11.5.2.1 Kendall's τ and τ_b	354
11.5.2.2 Spearman's Rank Correlation Coefficient ρ	355
11.5.3 Measures of Association Between Two Quantitative Variables	355
11.5.3.1 Covariance and Pearson's Correlation Coefficient	355
11.5.4 Measures of Association Between a Quantitative Variable and a Qualitative Variable	356
11.5.4.1 Correlation Ratio $\eta_{Y X}^2$	356
11.6 Graphical Representations	357
11.6.1 Plotting Qualitative Variables	357
11.6.1.1 Cross Chart	357
11.6.1.2 Bar Charts	359

Contents		xxiii
11.6.1.3 Pareto Chart	360	
11.6.1.4 Stacked Bar Chart	361	
11.6.1.5 Pie Chart	361	
11.6.2 Plotting Ordinal Variables	362	
11.6.2.1 Bar Chart with Cumulative Frequencies Line	362	
11.6.3 Plotting Discrete Quantitative Variables	363	
11.6.3.1 Cross Chart	363	
11.6.3.2 Bar Chart	363	
11.6.3.3 Plotting the Empirical Distribution Function	363	
11.6.3.4 Stemplot	365	
11.6.3.5 Boxplot	365	
11.6.4 Plotting Continuous Quantitative Variables	367	
11.6.4.1 Empirical Distribution Function	367	
11.6.4.2 Stemplot	367	
11.6.4.3 Boxplots	368	
11.6.4.4 Density Histogram with Identical or Different Class Ranges	368	
11.6.4.5 Frequency Polygon	369	
11.6.4.6 Cumulative Frequency Polygon	370	
11.6.5 Graphical Representations in a Bivariate Setting	371	
11.6.5.1 Two-Way Plots for Two Qualitative Variables	371	
11.6.5.2 Two-way Plots for Two Quantitative Variables	374	
11.6.5.3 Two-Way Plots for One Qualitative and One Quantitative Variable	375	
Memorandum	377	
Exercises	377	
Worksheet	378	
12 A Better Understanding of Random Variables, Distributions and Simulations Using R Specificities	381	
12.1 Notions on Random Number Generation	381	
12.2 The Notion of Random Variables	383	
12.2.1 Realizations of a Random Variable and Functioning Law	383	
12.2.2 I.i.d. Random Variables	384	
12.2.3 Characterizing the Distribution of a Random Variable	385	
12.2.3.1 Density Function, Distribution Function and Quantile Function	387	
12.2.4 Parameters of the Distribution of a Random Distribution	390	
12.3 Law of Large Numbers and Central Limit Theorem	392	
12.3.1 Law of Large Numbers	392	
12.3.2 Central Limit Theorem	393	

12.4	Inferential Statistics	394
12.4.1	Point Estimate of Parameters	394
12.4.2	Empirical Cumulative Distribution Function	396
12.4.3	Maximum Likelihood Estimation	397
12.4.4	Sampling Variation and Properties of an Estimator	399
12.5	A Few Techniques to Draw from a Distribution	401
12.5.1	Simulating from Another Distribution	401
12.5.2	Inverse Transform Method	402
12.5.3	Rejection Sampling	402
12.5.4	Simulation of Discrete Random Variables	403
12.6	Bootstrap	404
12.7	Standard and Less Standard Distributions	405
12.7.1	Standard Distributions	405
12.7.2	† Less Standard Distributions	408
12.8	Modelling a Phenomenon	410
	Memorandum	413
	Exercises	413
	Worksheet	413
13	Confidence Intervals and Hypothesis Testing	417
13.1	Notations	417
13.2	Confidence Intervals	418
13.2.1	Confidence Intervals for the Mean	418
13.2.2	Confidence Intervals for a Proportion p	419
13.2.3	Confidence Intervals for a Variance	421
13.2.4	Confidence Intervals for a Median	422
13.2.5	Confidence Intervals for a Correlation Coefficient	423
13.2.6	Summary Table for Confidence Intervals	424
13.3	Standard Hypothesis Testing	424
13.3.1	Parametric Tests	426
13.3.1.1	Tests of the Mean	426
13.3.1.2	Tests of Variance	429
13.3.1.3	Tests of Proportion	431
13.3.1.4	Tests of Correlation	433
13.3.2	Independence Tests	435
13.3.2.1	χ^2 Test for Independence	435
13.3.2.2	Yates' χ^2 Test	437
13.3.2.3	Fisher's Exact Test	438
13.3.3	Non-parametric Tests	439
13.3.3.1	Goodness-of-Fit Tests	439
13.3.3.2	Tests of Position	442
13.3.4	Memorandum of Standard Tests	447
13.4	Other Tests	447
	Memorandum	449
	Exercises	449
	Worksheet	449

14 Simple and Multiple Linear Regression	455
14.1 Introduction	455
14.2 Simple Linear Regression	456
14.2.1 Aim and Model	456
14.2.2 Fitting Data	457
14.2.3 Confidence and Prediction Intervals for a New Value	461
14.2.4 Analysis of Residuals	463
14.2.5 Student's Tests for Means and Linear Model	466
14.2.6 Summary	468
14.3 Multiple Linear Regression	468
14.3.1 Aim and Model	468
14.3.2 Fitting Data	469
14.3.3 Confidence and Prediction Intervals for a New Value	473
14.3.4 Testing a Linear Sub-hypothesis: Partial Fisher Test	473
14.3.5 Qualitative Variables with More Than Two Modalities	474
14.3.6 Interaction Between Variables	478
14.3.7 Issues with Collinearity	481
14.3.8 Variable Selection	482
14.3.9 Analysis of Residuals	490
14.3.10 Polynomial Regression	496
14.3.11 Summary	496
Memorandum	497
Exercises	497
Worksheet	498
15 Elementary Analysis of Variance	503
15.1 Analysis of Variance with One Factor	503
15.1.1 Aims, Data and Model	503
15.1.2 Example and Graphical Inspection	504
15.1.3 ANOVA Table and Parameter Estimation	505
15.1.4 Validation of Assumptions	509
15.1.5 Multiple Comparisons and Contrasts	510
15.1.6 Summary	512
15.2 Analysis of Variance with Two Factors	513
15.2.1 Aims, Data and Model	513
15.2.2 Example and Graphical Inspection	514
15.2.3 ANOVA Table, Tests and Parameter Estimation	516
15.2.4 Validating Assumptions	519
15.2.5 Contrasts	519
15.2.6 Summary	521
15.3 Repeated Measures Analysis of Variance	521
15.3.1 One-Way Repeated Measures ANOVA	522
15.3.2 Two-Factor Model with Repeated Measures for Both Factors	523

15.3.3 Two-Factor Model with Repeated Measures for One Factor	525
Memorandum	527
Exercises	527
Worksheet	527
Appendix: Installing R and R Packages	531
A.1 Installing R Under Microsoft Windows	531
A.2 Installing Additional Packages	532
A.2.1 Installing from a File on Your Disk	532
A.2.2 Installing Directly from the Internet	533
A.2.3 Installing from the Command Line	535
A.2.4 Installing Packages Under Linux	535
A.3 Loading Installed Packages	536
References	539
General Index	541
Index of R Commands and Symbols	549
Index of Authors	561
List of R Packages Mentioned in the Book	563
Solutions to Exercises	565
Solutions to Worksheet	577

List of Figures

1.1	A few of the graphical possibilities offered by R	6
1.2	The RCommander graphical interface	9
1.3	Entering data with the RCommander graphical interface	10
1.4	Basic statistics with RCommander	11
1.5	Manipulating a data set with RCommander	12
1.6	Mean comparison test with RCommander	15
1.7	Independence test with RCommander	16
1.8	Least squares plane	18
3.1	The script window and the command console	41
3.2	Characteristics of a complex number	47
3.3	Illustration of an array	53
7.1	Effect of argument <code>mfrw</code> of function <code>par()</code> . Numbers have been added to gain a better understanding of where future plots will be drawn	154
7.2	Potential of the function <code>layout()</code>	155
7.3	The function <code>layout()</code> and its arguments <code>widths</code> and <code>heights</code>	156
7.4	The function <code>plot()</code>	157
7.5	The function <code>points()</code>	158
7.6	The functions <code>segments()</code> and <code>lines()</code>	159
7.7	The function <code>abline()</code>	159
7.8	The function <code>arrows()</code>	161
7.9	The function <code>curve()</code>	162
7.10	The function <code>box()</code>	163
7.11	The argument <code>col</code> of function <code>plot()</code>	164
7.12	The argument <code>alpha</code> of function <code>rgb()</code>	165
7.13	An example using function <code>rainbow()</code>	166
7.14	The function <code>display.brewer.all()</code>	167
7.15	The function <code>image()</code>	168
7.16	The function <code>image()</code> with a coherent display of the data	168

7.17	The function <code>text()</code>	169
7.18	The function <code>mtext()</code>	170
7.19	The function <code>title()</code>	171
7.20	Plot title on several lines	172
7.21	The function <code>axis()</code>	173
7.22	The function <code>legend()</code> with squares	174
7.23	The function <code>legend()</code> with line segments	174
7.24	Figure illustrating the fine management of graphical parameters	178
7.25	Managing the colours of a plot	179
7.26	Example of use of the arguments <code>adj</code> and <code>srt</code>	181
7.27	Using different fonts on a plot	182
7.28	Labels on a plot	184
7.29	The arguments <code>lend</code> and <code>ljoin</code>	185
7.30	The argument <code>pch</code>	186
7.31	The arguments <code>lty</code> and <code>lwd</code>	186
8.1	Result of the call of the function <code>mydisplay.reg1()</code>	211
8.2	Emacs and GDB	263
8.3	DDD and GDB	265
9.1	Illustration of storage of values in memory. Each little box contains a binary number (0 or 1). Each green number gives the decimal representation of the number in binary form in the block above. Each red number gives the address (expressed here in decimal notation) of the 8-box block above. Note that the same memory addresses could have been written in hexadecimal notation ($b = 16$), giving 3C, 3D, 3E and 3F	294
9.2	Illustration of R storage in memory of a (signed) integer. Each little box contains a binary digit (0 or 1). The green number gives the decimal notation of the integer expressed in binary notation in the four blocks above. The red number gives the address (expressed here in decimal base) of the first 8-box memory block above. Note that here, a number is stored over 32 boxes and not over 8 as in Fig. 9.1. Furthermore, the first box is used to specify the sign of the number, negative here	295
10.1	Modified sinc function	329
11.1	Algorithm to determine the type of a variable	340
11.2	Cross chart for a qualitative variable	358
11.3	Dot chart for a qualitative variable	358
11.4	Bar chart for a qualitative variable	359
11.5	Pareto chart for a qualitative variable	360
11.6	Stacked bar chart for a qualitative variable	361
11.7	Bar chart with cumulative frequencies line for an ordinal variable	363

11.8	Bar chart for a discrete quantitative variable	364
11.9	Empirical distribution function for a discrete quantitative variable..	364
11.10	Boxplot and explanations	366
11.11	Plot of the empirical distribution function for a continuous quantitative variable.	367
11.12	Density histogram with identical or different class ranges	369
11.13	Frequency polygon	370
11.14	Cumulative frequency polygon	371
11.15	Bar plot for two qualitative variables	372
11.16	Mosaic plot for two qualitative variables	372
11.17	Cohen–Friendly association plot for two qualitative variables	373
11.18	<code>table.cont</code> plot for two qualitative variables	374
11.19	Plot of two quantitative variables.....	375
11.20	Box plots of a quantitative variable as a function of the levels of a qualitative variable	376
11.21	<code>stripchart</code> plot for a quantitative and a qualitative variable.	376
12.1	Plot approximating the density of X	388
12.2	Convergence in distribution in action on an example with simulated data.....	394
14.1	Scatter plot of child weight against mother weight.....	457
14.2	Least squares regression line on a scatter plot.....	458
14.3	Visualization of confidence and prediction intervals	463
14.4	Graphical inspection of normality of residuals	464
14.5	Plot of residuals as a function of predicted values	465
14.6	Scatter plot of all pairs of variables	470
14.7	Effect of age on BWT in a model without interaction	479
14.8	Effect of age on BWT in a model with interaction	480
14.9	Selecting variables with the BIC	484
14.10	Checking the assumptions of homoscedasticity (<i>left</i>) and normality (<i>right</i>)	490
14.11	Residuals as a function of explanatory variables	491
14.12	Visualizing outliers: studentized residuals against fitted values	492
14.13	Visualizing influential observations: Cook's distance	494
15.1	Box plot of scarring times for each treatment.....	506
15.2	Analysing the residuals in single-factor ANOVA	509
15.3	Exploration of interaction in two-way ANOVA	516
15.4	Residual analysis in two-way ANOVA	520

List of Tables

3.1	The various data types in R	50
3.2	The various data structures in R	58
4.1	Data importation functions	64
4.2	Main arguments to <code>read.table()</code>	64
4.3	Packages and R importation functions from common software	71
5.1	Operators and functions which take logical values as input or output	98
5.2	Operations on sets	99
5.3	Codes for the function <code>strptime()</code>	113
5.4	Correspondence between BMI and weight categories	124
7.1	Parameters to manage the graphics window.....	177
7.2	Colour management parameters	179
7.3	Managing text displayed on a plot	180
7.4	Parameters to manage axes	183
7.5	Parameters for lines and symbols	185
8.1	Conventions on argument types	238
10.1	Table of basic mathematical functions	314
12.1	Standard discrete distributions	405
12.2	Standard continuous distributions	406
12.3	Less standard distributions I	408
12.4	Less standard distributions II	409
13.1	Some notation for standard parameter estimation	417
13.2	Notation of various quantiles of order p	418
13.3	Summary for confidence intervals	424
13.4	Standard tests	447

14.1 Main R functions for simple linear regression	468
14.2 Main R functions for multiple linear regression	496
15.1 Main functions for single-factor ANOVA	513
15.2 Main functions for two-way ANOVA	521

Mathematical Notations

$:=$	Symbol indicating different notations for a single object
\cup	Table fusion
$a \in A$	a belongs to set A
$A \subset B$	A is included in B
$A \supset B$	A includes B
$A \cap B$	Intersection of sets A and B
$A \cup B$	Union of sets A and B
$A \setminus B$	Complement of set B in set A
$(A \cup B) \setminus (A \cap B)$	Symmetric difference of sets A and B
f_i	Frequency of a modality
$ x $	Absolute value of number x
$x!$	Factorial of number x
$\binom{n}{p}$	Binomial coefficient: number of ways of picking p elements out of n
$\Gamma(\cdot)$	Gamma function
γ	Euler's constant
$\psi(\cdot)$	Digamma function
π	Number π
λ	Scalar number
$\mathcal{A}, \mathcal{B}, \mathcal{C} \dots$	Matrices
\mathcal{I}	Identity matrix
$n \times p$	Indicates the size of a matrix
\mathcal{A}^T	Transpose of matrix \mathcal{A}
\mathcal{B}^{-1}	Inverse of matrix \mathcal{B}
$\bar{\mathcal{C}}$	Conjugate of complex matrix \mathcal{C}
$\mathbf{x} = (x_1, \dots, x_n)^T$	Column vector
\mathbf{x}^T	Transpose of vector \mathbf{x}
$\mathcal{A} \otimes \mathcal{B}$	Kronecker product of matrix \mathcal{A} with matrix \mathcal{B}
$\text{vec}(\mathcal{A})$	Vector resulting from the stacking of columns of matrix \mathcal{A}
$\text{vech}(\mathcal{A})$	Vector resulting from the stacking of columns of matrix \mathcal{A} , but excluding elements above the diagonal

\mathcal{A}^*	Adjunct matrix (conjugate transpose) of matrix \mathcal{A}
$\mathcal{A}^{1/2}$	Square root of matrix \mathcal{A}
$\mathbb{1}_{[A]}(x)$	Equals 1 if $x \in A$ and 0 otherwise
$[a, b]$	Interval of values between a and b
$\det(\mathcal{A})$	Determinant of matrix \mathcal{A}
$\Phi(\cdot)$	Cumulative distribution function of a standard normal random variable $\mathcal{N}(0, 1)$
$\dot{\mathcal{X}}$	Matrix given by centring the columns of matrix \mathcal{X}
$\mathbb{1}_n$	Vector $(1, \dots, 1)^T$ of length n
X, Y	Non-random variables (descriptive statistics)
N	Population size
n	Sample size
$m_e := q_{1/2}$	Median
$PFC_X(\cdot)$	Value of the polygon of cumulative frequencies of X
μ_X	Expected value of random variable X , or population mean in descriptive statistics
q_p or x_p	Fractile (quantile) of order p of a variable
$q_{1/4}, q_{3/4}$	First and third quartiles (also noted q_1 and q_3)
$\sigma_{Pop}^2(\mathbf{x})$	Population variance (descriptive statistics)
$\sigma_{Pop}(\mathbf{x})$	Population standard error (descriptive statistics)
c_v	Population coefficient of variation (descriptive statistics)
γ_1	Skewness
β_2	Kurtosis
μ_3	Centred moment of order 3
μ_4	Centred moment of order 4
χ^2	Pearson's χ^2 statistic
Φ^2, V^2	Cramér's Φ^2 and V^2
τ, τ_b	Kendall's τ and τ_b
ρ	Theoretical Pearson coefficient of correlation
$\eta_{Y X}^2$	Correlation ratio
X, Y, ϵ	Random variables
x_i, y_i, ϵ_i	Realizations of random variables X, Y, ϵ
$\mathbf{X}, \mathbf{Y}, \boldsymbol{\epsilon}$	Random vectors
\mathbf{x}_n	Sample (random)
\mathbf{x}_n	Sample (observed)
\mathbf{X}	Random matrix
\mathcal{L}	Generic distribution of a random variable
$\mathcal{N}(0, 1)$	Standard normal distribution
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$\mathcal{U}(a, b)$	Uniform distribution over the interval $[a, b]$
$\mathcal{B}in(n, p)$	Binomial distribution with parameters n and p
$\mathcal{E}(\lambda)$	Exponential distribution with parameter λ

$\mathcal{P}(\lambda)$	Poisson distribution with parameter λ
$\mathcal{T}(n)$	Student distribution with n degrees of freedom
$\chi^2(n)$ or χ_n^2	χ^2 distribution with n degrees of freedom
$\mathcal{F}(n, m)$	Fisher distribution with n and m degrees of freedom
$f_X(\cdot)$	Probability density function of random variable X
$F_X(\cdot)$	Cumulative distribution function of random variable X
$F_X^{-1}(\cdot)$	Inverse cumulative distribution function of random variable X
μ	Expected value of a random variable
σ^2	Variance of a random variable
$\mathbb{E}(Y)$	Theoretical expectation of random variable Y
$\text{Var}(Y)$	Theoretical variance of random variable Y
\bar{X}_n	Empirical mean $\frac{1}{n} \sum_{i=1}^n X_i$ of sample $\mathbf{X}_n = (X_1, \dots, X_n)^\top$, estimator of μ_X
\bar{x}_n	Realization of the empirical mean $\frac{1}{n} \sum_{i=1}^n X_i$ of sample $\mathbf{X}_n = (x_1, \dots, x_n)^\top$, estimate of μ_X
\xrightarrow{P}	Convergence in probability
$\hat{F}_n(\cdot) := \hat{F}_{\mathbf{X}_n}(\cdot)$	Empirical cumulative distribution function of sample \mathbf{X}_n
θ	Unknown parameter (the true unknown value of the parameter will sometimes be noted θ^*)
$\hat{\theta}(x_1, \dots, x_n)$ or $\hat{\theta}$	Estimator of unknown parameter θ based on the sample $\mathbf{X}_n = (x_1, \dots, x_n)^\top$
$\hat{\theta}(x_1, \dots, x_n)$ or $\hat{\theta}$	Estimate of unknown parameter θ based on the observed sample $\mathbf{x}_n = (x_1, \dots, x_n)^\top$
$\mathbb{B}(\hat{\theta}(x_1, \dots, x_n); \theta)$	Bias of estimator $\hat{\theta}(x_1, \dots, x_n)$ to estimate unknown parameter θ
$P[A]$	Probability of set A
$\mathcal{V}(\theta; X_1, \dots, X_n)$	Likelihood function of sample \mathbf{X}_n evaluated at θ
$\mathbf{x}^* = (x_1^*, \dots, x_n^*)^\top$	Bootstrap sample generated from the observed sample $\mathbf{x}_n = (x_1, \dots, x_n)^\top$
$\hat{\sigma}$	Estimator of σ
$\hat{\sigma}$	Estimate of σ
p	Proportion
\hat{p}	Estimator of a proportion (or of a probability)
\hat{p}	Estimate of a proportion (or of a probability)
\widehat{m}_e	Estimator of a median
\widehat{m}_e	Estimate of a median
M	Number of iterations (of generated samples) in a Monte Carlo simulation
B	Number of generated bootstrap samples
$B(\cdot, \cdot), \Gamma(\cdot)$	Beta function, gamma function
$I'_x(\cdot, \cdot)$	Derivative of incomplete beta function

$I(\cdot)$	Modified Bessel function
$I_\alpha(\cdot)$	Modified Bessel functions
u_p	Quantile of order p of a $\mathcal{N}(0, 1)$
t_p^n	Quantile of order p of a $\mathcal{T}(n)$
q_p^n	Quantile of order p of a $\chi^2(n)$
$f_p^{n,m}$	Quantile of order p of a $\mathcal{F}(n, m)$
$CI_{1-\alpha}(\theta)$	Random confidence interval at confidence level $1 - \alpha$ for θ
$ci_{1-\alpha}(\theta)$	Realized confidence interval at confidence level $1 - \alpha$ for θ
$1 - \alpha$	Level of a confidence interval
$(x_{(1)}, \dots, x_{(n)})$	Observed sample, sorted from smallest to largest value
\mathcal{H}_1	Assertion of interest in hypothesis testing
\mathcal{H}_0	“Null” hypothesis, opposite of \mathcal{H}_1
α	Significance level or risk of the first kind in hypothesis testing
R	Random Pearson empirical coefficient of correlation
r	Realized Pearson empirical coefficient of correlation
β_0, β_1	Unknown coefficients of a simple linear regression model
$\hat{\beta}_0, \hat{\beta}_1$	Estimates of unknown coefficients of a simple linear regression model
$\hat{\epsilon}_i$	Observed residuals in a simple linear regression model
\hat{y}_i	Adjusted observed values in a simple linear regression model
R^2	Random coefficient of determination in regression
r^2	Realized coefficient of determination in regression
R_a^2	Random adjusted coefficient of determination in regression
r_a^2	Realized adjusted coefficient of determination in regression
\hat{Y}^p	Predictor of random variable Y for a new value of the explanatory variable X in regression
$PI_{1-\alpha}(Y_0, x_0)$	Prediction interval at level $1 - \alpha$ for random variable Y_0 associated with a new value x_0 of the explanatory variable
$\boldsymbol{\beta} = (\beta_0, \dots, \beta_p)^\top$	Vector of the $p + 1$ unknown coefficients in a multiple linear regression model with p explanatory variables
$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$	Estimator of the vector of unknown parameters $\boldsymbol{\beta}$ for the matrix \mathbf{X} of observed explanatory variables and for the observed vector of explained values in a muliple linear regression model
$\hat{\boldsymbol{\beta}}$	Estimate of $\boldsymbol{\beta}$
VIF	Variance inflation factor
AIC	An Information Criterion
BIC	Bayesian information criterion
h_i	Leverage of i th observation in regression
t_i	Standardized residuals
t_i^*	Studentized residuals
$\hat{\sigma}_{(-i)}$	Estimate of σ excluding i th observation

C_i	Cook's distances
$\hat{y}_j^{(-i)}$	Prediction of y_j , not using the i th observation
$\hat{\beta}_j^{(-i)}$	Estimate of β_j , not using the i th observation
I, J	Number of levels of a factor in ANOVA
$\mu_{\bullet\bullet}$	Mean general effect in ANOVA
$\mu_{i\bullet}$	Effect of level i of a factor in ANOVA
$\mu_{\bullet j}$	Effect of level j of a factor in ANOVA

Part I

Preliminaries

Chapter 1

Introducing R

Prerequisites and goals of this chapter

- You may find it useful to read the chapter on installing R in the Appendix first.
- This chapter presents the origins, objectives and specificities of R.

— SECTION 1.1 —

Presentation of the Software

1.1.1 Origins

R is a piece of statistical software created by Ross Ihaka and Robert Gentleman [21]. R is both a programming language and a work environment. Commands are executed using descriptive code. Results are displayed as text and the plots are visualized directly in their own window. R is clone of the statistical software S-plus. S-plus is an object-oriented programming language S developed by AT&T Bell Laboratories in 1988 [3]. S-plus is used to manipulate data, draw plots and perform statistical analyses of data.

1.1.2 Why Use R?

First of all, R is **free** and **open-source**. It works under UNIX (and Linux), Microsoft Windows and Macintosh Mac OS: it is **cross-platform**. It is being developed in the

free software movement by a large and growing community of eager volunteers. Anyone can contribute to and improve R by integrating more functionalities or analysis methods. It is thus a quickly and constantly evolving piece of software.

R is a very powerful statistical tool. The learning curve in R is steeper than other statistical software on the market such as SPSS, SAS or Minitab. R is not the kind of statistical package, which you can use with a few clicks of the mouse in the menus. In order to use it, you need to understand the statistical method that you are trying to implement, so R is a didactic program. R is also very efficient and easy to implement once you have mastered it. You will be able to create your own tools and you will be able to handle and work on very sophisticated data analyses.

Warning



R is harder to comprehend than other software on the market. You need to spend time learning the syntax and commands.

R is especially powerful for data manipulation, calculations and plots. Its features include:

- an integrated and very well-conceived documentation system (in English)
- Efficient procedures for data treatment and storage;
- a suite of operators for calculations on tables, especially matrices;
- a vast and coherent collection of statistical procedures for data analysis;
- advanced graphical capabilities;
- a simple and efficient programming language, including conditioning, loops, recursion, and input-output possibilities.

Note



For the readers already used to SAS, SPSS or Stata, we advise to read the books [32, 33] and also to consult the two following Internet websites:

- <http://rforsasandspssusers.com>
- <http://www.statmethods.net>

Note also that it is possible to call R functions directly from Matlab using the R.matlab package and from Excel using the RExcelInstaller package. Reading of [20] might be useful in this context. Finally, a similar tool for OpenOffice, called ROOo, exists; see the Internet website <http://rcom.univie.ac.at>.

SECTION 1.2

R and Statistics

Many classical and modern statistical techniques are implemented in R. The most common methods for statistical analysis, such as

- descriptive statistics;
- hypothesis testing;
- analysis of variance;
- linear regression methods (simple and multiple)

are directly included at the core of the system. It should be noted that most advanced statistical methods are also available through external packages. These are easy to install, directly from a menu. They are all grouped and can be browsed on the website of the *comprehensive R archive network* (CRAN) (<http://cran.r-project.org>). This website also includes, for some large domains of interest, a commented list of packages associated with a theme (called Task View). This facilitates the search for a package on a specific statistical method. Furthermore, detailed documentation for each package is available on the CRAN.

It should also be noted that recent statistical methods are added on a regular basis by the statistics community itself.

See also

Section A.2, p. 532, gives details on the procedure to install a new package.



SECTION 1.3

R and Plots

One of the main strengths of R is its capacity (much greater than that of other software on the market) to combine a programming language with the ability to draw high-quality plots. Usual plots are easily drawn using predefined functions. These functions also include many parameters, for example to add titles, captions and colours. But it is also possible to create more sophisticated plots to represent complex data such as contour lines, volumes with a 3D effect, density curves, and many other things. It is also possible to add mathematical formulae. You can arrange or overlay several plots in the same window and use many colour palettes.

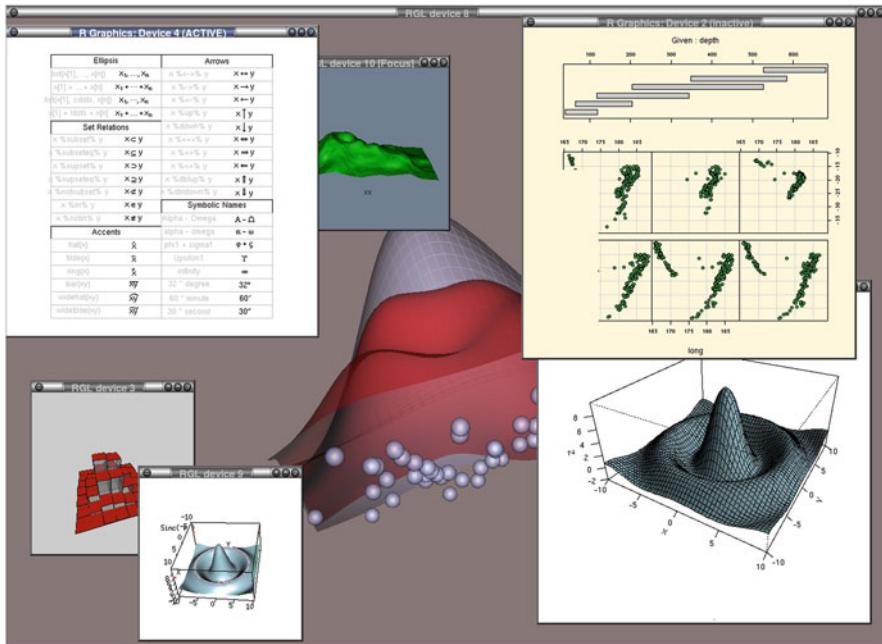


Fig. 1.1: A few of the graphical possibilities offered by R

You can get a demonstration of the graphical possibilities in R by typing in the following instructions:

```

demo(image)
example(contour)
demo(graphics)
demo(persp)
demo(plotmath)
demo(Hershey)
require("lattice") # Load the package, which you must have
                  # previously installed by using the menu
                  # Packages/Install packages.

demo(lattice)
example(wireframe)
require("rgl")      # Same remark as above.
demo(rgl)          # You can interact by using your mouse.
example(persp3d)

```

Figure 1.1 above shows a few of these plots.

— SECTION 1.4 —

The R Graphical User Interface

The R graphical user interface (GUI) (i.e. its set of menus) is very limited, and completely nonexistent on some operating systems, when compared to other standard software. This minimality can set back some new users. However, this drawback is limited since:

- it has the didactic advantage that it incites users to know well the statistical procedures they wish to use;
- there are additional tools which extend the GUI.

In the next section, we present the package `Rcmdr`, which can be installed from the menu `Packages` and which allows standard graphical and statistical analyses with a more user-friendly interface, which includes drop-down menus. Furthermore, the R instructions for the analysis chosen from the `RCommander` menus are displayed in dedicated panel. This can be useful if you do not know (or remember) the R instruction for a specific task.

Tip

Note that after you have learnt R thoroughly, you will be able to develop yourself tools similar to `Rcmdr`, made for a final users who do not desire to learn R but only to use, in the most user-friendly way, a procedure created by you. To this end, you can use the package `tcltk`.

**Warning**

Note that by using `RCommander`, we are distancing ourselves from what makes the strength and flexibility of R. We therefore advise against using such a tool if you wish to become an advanced user.



— SECTION 1.5 —

First Steps in R

1.5.1 Using RCommander

In this section, we offer a brief introduction to the package `Rcmdr`. We then present some functionalities given by this interface for statistical manipulations. We conclude by explaining how to add functionalities to the `RCommander` interface.

1.5.1.1 Launching RCommander

Follow these steps to start RCommander.

- ▶ Double-click on the R icon on your Desktop.
- ▶ In the console, type `install.packages("Rcmdr")`. Choose a nearby mirror.
- ▶ In the console, type `require("Rcmdr")`. Answer Yes to all the questions you may be asked. The RCommander graphical interface then opens. Another option is to click on the menu Packages, then Load package..., then Rcmdr.
- ▶ In the Messages panel, you should see **WARNING: the Windows version of R Commander works better under RGui with the single document interface (SDI)**.
- ▶ To remedy this issue, close RCommander.
- ▶ In RGui, go to Edit, then Preferences. Check SDI then click on Save... and on Save. You can take this opportunity to customize R.
- ▶ Close R and save an image of the session.
- ▶ Restart R, then RCommander by typing `require("Rcmdr")` in the R console.

See also



We refer the reader to Sect. A.2 which details how to install the package Rcmdr.



Mac

Macintosh users may find useful the instructions at <http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html>, after installing package tcltk which is available on the CRAN.

The graphical interface of RCommander includes four parts as shown on Fig. 1.2:

- (a) Drop-down menus to perform specific tasks
- (b) A Script window which presents the code executed thanks to click on a drop-down menu
- (c) An Output window which gives the output of the executed code
- (d) A Messages window giving a message on the last task

1.5.1.2 Handling Data with RCommander

To perform statistical analyses, you need data.

- **Entering data by hand**

Follow these steps to enter data by hand.



Fig. 1.2: The RCommander graphical interface

- ▶ In the menu Data, choose New data set....
- ▶ In the window New data table, choose a name for your data set, for example Data1.
- ▶ A data editor appears. Click on var1 and replace it with Name. Enter a few names for this variable: Peter, Jack, Ben (see Fig. 1.3).
- ▶ Create a variable Height of type numeric with the following values: 182, 184, 190.
- ▶ Click on the cross (X) at the top-right corner of the active window to close the data editor.
- ▶ You can visualize your data set by clicking on View.

We can now calculate some basic statistics.

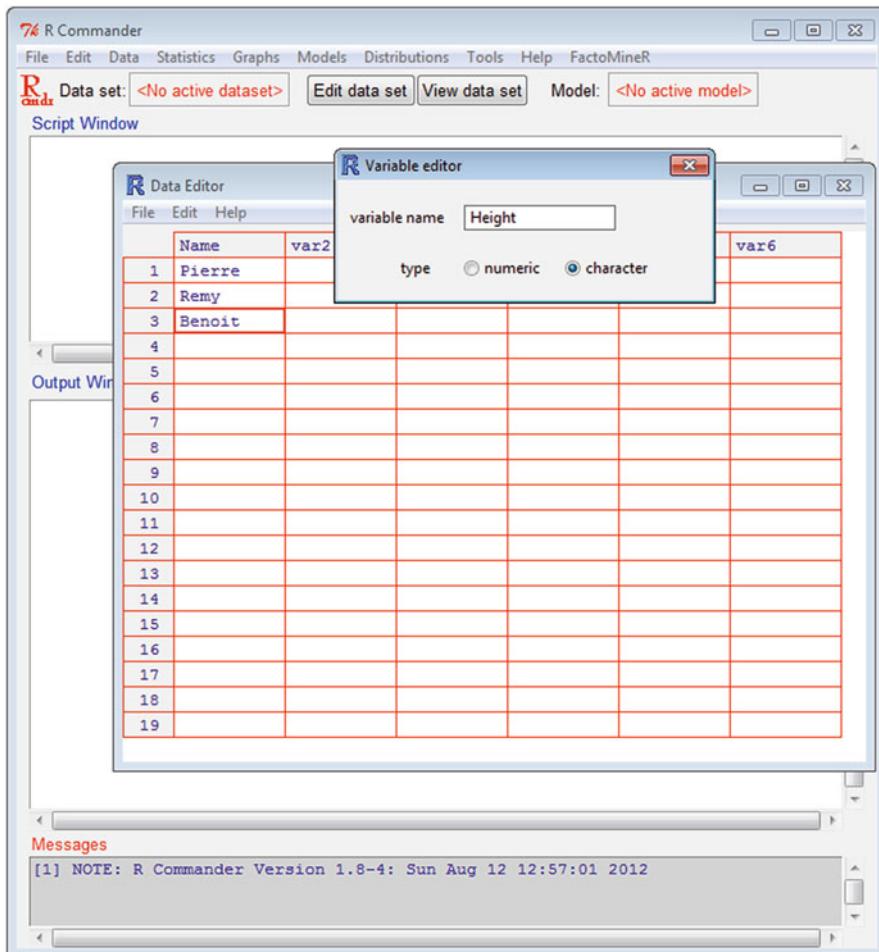


Fig. 1.3: Entering data with the RCommander graphical interface

• Basic statistics

Follow these steps to get some basic statistics on your data set:

- In the menu **Statistics**, choose **Summary**, then **Descriptive statistics**
- A window called **General statistics** opens up; the only **numeric** variable in our data set is the variable **Height**.
- Choose the statistics **Mean**, **Standard deviation** and **Quantiles** and click on **OK**.
- The result is displayed in the **Output window**. Note that you can check the **R** instruction which was used for this task in the **Script window** (see Fig. 1.4).

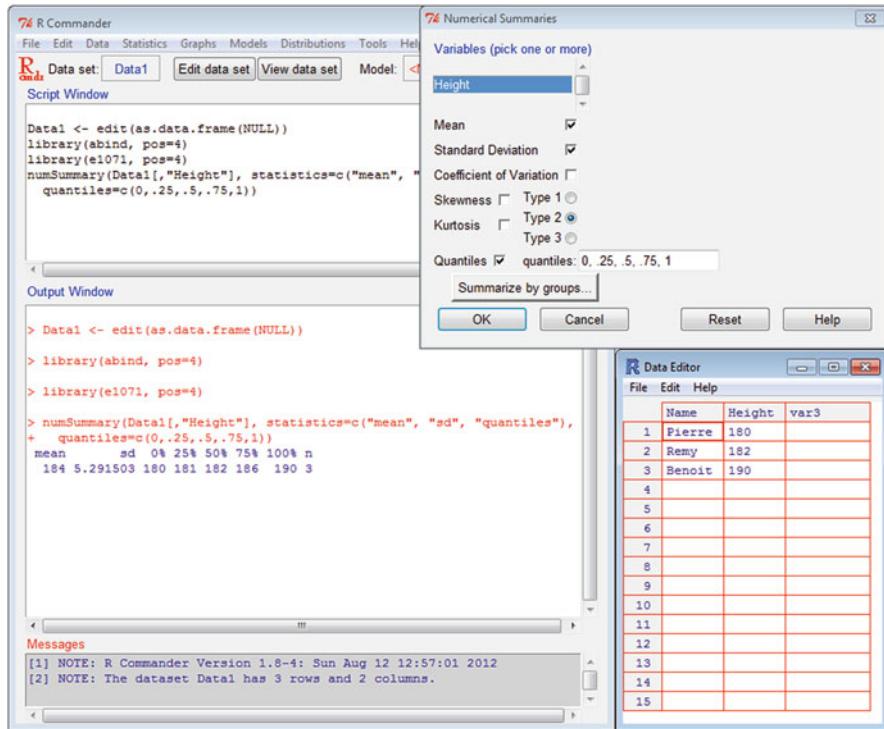


Fig. 1.4: Basic statistics with RCommander

Note that it is also possible to type an instruction directly in the **Script window** without using a menu. Here is an example.

► Type in the **Script window**:

```
numSummary(Data1[, "Height"], statistics=c("mean", "sd"))
```

- Click on that line so that the cursor is displayed there, then click on **Submit**.
- You have just computed the mean and standard deviation of variable **Height** which contains 3 observations. The result appears in the **Output window**:

```
mean      sd % n
184 5.291503 0 3
```

• **Manipulating the data set**

In our toy example, suppose that we also have the weight and wish to compute the body mass index: $BMI = Weight/Height^2$ (height in metres).

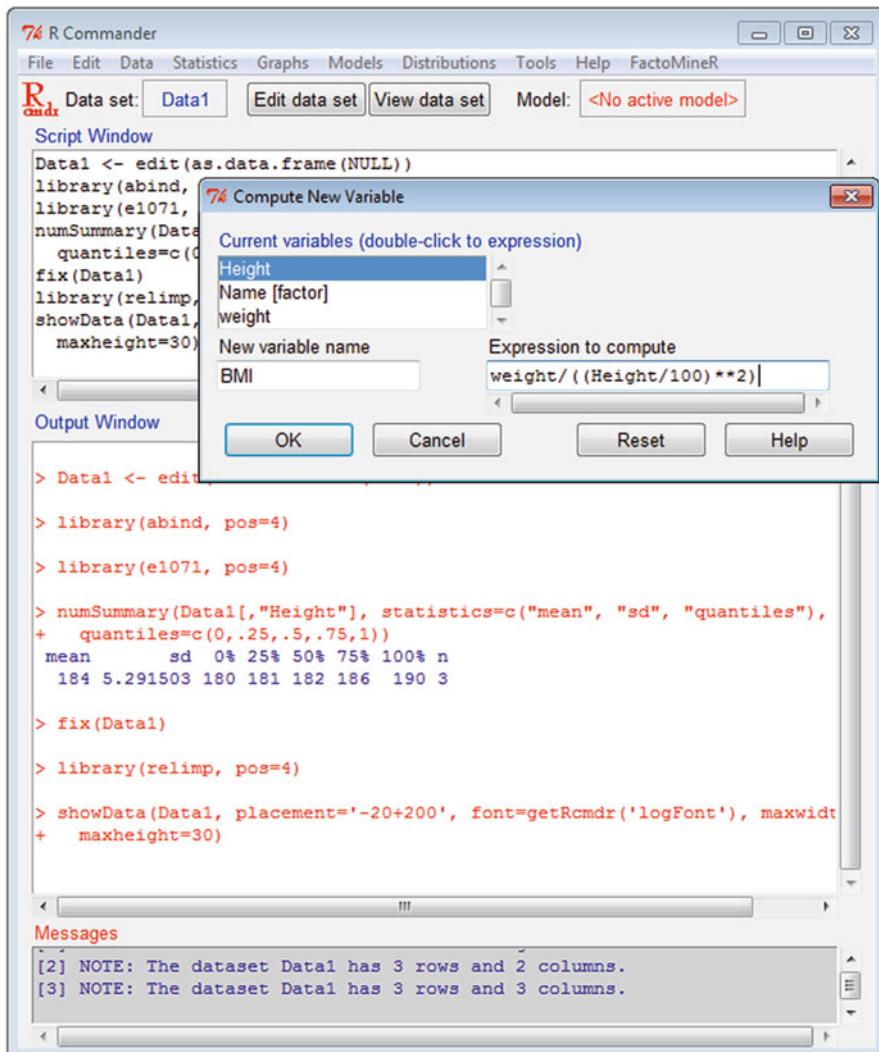


Fig. 1.5: Manipulating a data set with RCommander

- ▶ Click on Edit (below the RCommander menus).
- ▶ The data editor opens up and you can add the numeric variable Weight, with the following values: 70, 72 and 75. Now close the data editor.
- ▶ In the Data menu, choose Manage variables in the active data set, then Calculate a new variable.... A window opens.
- ▶ For Name of new variable, type BMI and for Expression to calculate: Weight / ((Height/100)**2) (see Fig. 1.5). Click on OK to complete the calculation.
- ▶ Click on View to see the result for your data set.

You are starting to feel tired and need a coffee break! But before you take one, follow these steps to save your data set.

- ▶ In the Data menu, choose Active dataset, then Save active dataset....
- ▶ A window opens. You can choose a location to save your data set. We shall call it BMI and by default it has the .RData extension.
- ▶ Close RCommander and answer OK to the question Do you wish to quit?, No to Save script file? and No to Save output file?.
- ▶ You can now close R and answer No to the question Save session image?.

After a well-deserved break, you wish to add new data to your file BMI.RData.

- ▶ Open an R session. Type library("Rcmdr").
- ▶ In the Data menu, choose Load data set....
- ▶ A window opens. Navigate to and open the file BMI.RData.
- ▶ Click on View to display your data set.
- ▶ Add the information for a new person ("Julia", Height=150, Weight=52) by clicking on Edit.
- ▶ After closing the editor, you can check the changes by clicking on View. You then see the value NA (*not available*) for Julia's BMI.
- ▶ To get Julia's BMI, you need to go through the steps of section *manipulating the data set* again. We shall see later on how to create a function which calculates the BMI in a more user-friendly fashion.

You now wish to send your data set to a colleague who does not use R yet.

- ▶ In the Data menu, choose Active dataset, then Export active data set
- ▶ A first window opens. Uncheck the box Write names of individuals (rows) since we have not defined these. Choose Spaces for the field separator.
- ▶ Click on OK. A second window opens. You can choose a place to save your data set. We shall call it BMI and it has the default extension .txt.
- ▶ You can now send your data set BMI.txt to your colleague and use this opportunity to mention the wonderfulness of R, which has a rather user-friendly interface for data manipulation.

1.5.1.3 A Few Statistical Tasks with RCommander

In this section, we present a brief overview of how to use RCommander for statistical tasks. We start with a mean comparison test and a chi-square test of independence. We then show how to use RCommander to visualize the standard distributions (binomial, poisson, normal, gamma, etc.). We conclude with a linear model fit.

• Mean comparison test

We propose to use data already available in R. Follow these steps to load a data set:

- In the **Data** menu, choose **Data in packages**, then **Read data from an attached package....**
- A window opens. Double-click on **datasets** in the **Package** section, then on **sleep** in the right column.
- **sleep** appears in the box **Enter a dataset name** (see Fig. 1.6).
- You can now click on **Help on the selected dataset** to have some information about it.
- Click on **OK** to close the previous window, then visualize the data set by clicking on **View**.

These data are used to compare the effect on sleep of a soporific drug, compared to a control group. We shall first visualize the distribution of sleep gain in both groups, then do a mean comparison test to see whether there is any statistical significant difference between the drug and the control.

- In the **Graphs** menu, choose **Box plot....**
- A window opens. Click on **Plot by group....**, then on the **variable group**, then on **OK** twice.
- You can now see two box plots representing the sleep time gain in both groups.
- You can save this plot by clicking on **File**, then **Save as**. Several formats are possible.

You can also enhance this plot, for example, by adding colours. In the script window, type

```
boxplot(extra~group,ylab="extra",xlab="group",data=sleep,  
       col=c("red","blue"))
```

then click on **Submit**.

See also

Chapter 7 is dedicated to plots in R.

We now perform a mean comparison test.

- In the **Statistics** menu, choose **Means**, then **independent t-test....**
- Click on **group** in section **Groups (one)**. You now see specified the difference **1-2 (group 1 vs. group 2)**.
- Click on **OK** to see the result in the **Output window** (see also Fig. 1.6).

The *p*-value of this test (greater than 5 %) does not allow us to conclude that there is a significant difference between the sleep gains given by the drug and the control.

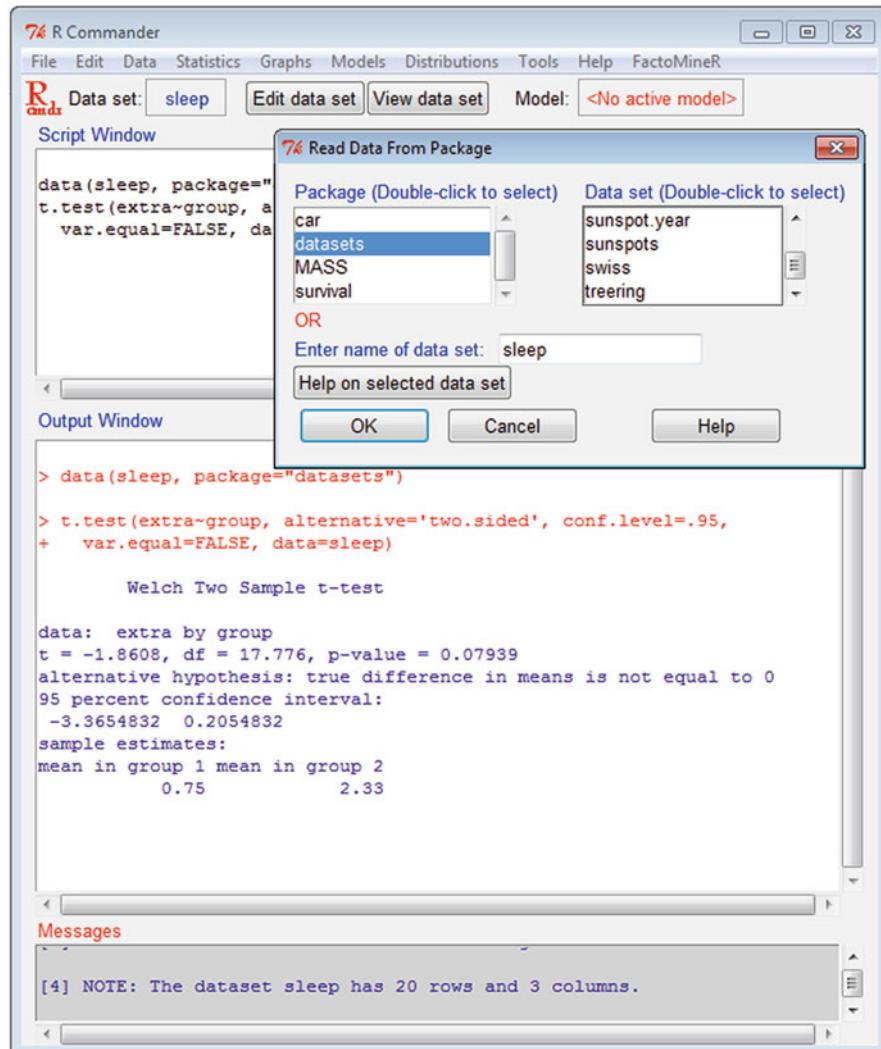


Fig. 1.6: Mean comparison test with RCommander

- **Test on a double entry table**

In a therapeutic test, the underlying question is whether a treatment on HIV-positive mothers has an effect on the HIV status of the child. If it does not, then the HIV status of the child is independent of the treatment taken by the mother. In this test, out of 391 children, 100 are HIV negative, 193 have mothers under treatment and 41 are HIV positive and have mothers under treatment. To know whether the treatment has an effect, follow these steps:

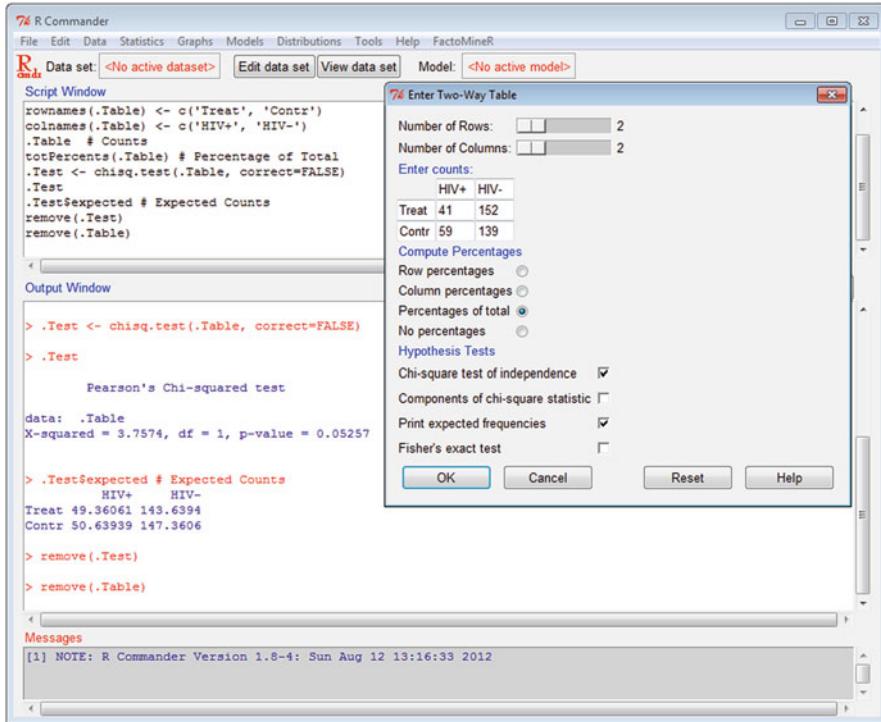


Fig. 1.7: Independence test with RCommander

- ▶ In the Statistics menu, choose Contingency tables, then Fill and analyse a double entry table....
- ▶ A window opens. Fill the table as indicated in Fig. 1.7. Choose Total percentages and Print expected frequencies.
- ▶ Click on OK to see the result in the Output window.

At the 5 % risk level, we cannot conclude that the treatment has an effect on the child's HIV status.

• Exploring distributions

RCommander can be used to visualize standard distributions.

- ▶ In the Distributions menu, choose Continuous distributions, the Normal distribution, then Plot of normal distribution....
- ▶ A window opens. Specify a mean of 4 and a standard deviation of 2. Click on OK.
- ▶ The curve of the density of a normal distribution centred at 4 and with standard deviation 2 appears in a graphical window.

You can follow the same steps for other probability distributions.

• Fitting a linear model

RCommander can be used to easily fit standard regression models. We illustrate this with the linear model. We shall first download a data set from an Internet address (URL). It contains the measures, for 80 patients with a disabling illness, of the variables GENDER (1 = Male, 2 = Female), WEIGHT (in kg), HEIGHT (in cm), PAIN (ordinal variable: a=least pain), DISTANCE (number of metres walked), MOBILITY (self-evaluation of mobility; 1=most mobile) and STAIRS (number of steps climbed).

- ▶ In the Data menu, choose Import data, then from a text file, the clipboard or a URL....
- ▶ A window opens. Call the data table Illness. Check the box Internet link (URL) in Data file and the box Tabulations for Field separator; click on OK.
- ▶ In the field Internet link (URL), type <http://biostatisticien.eu/springer/illness.txt>.
- ▶ Click on OK and you should see the following in the Messages window: The illness data set contains 80 rows and 8 columns.

We shall fit a multiple regression model. Follow these steps.

- ▶ In the Statistics menu, choose Model fitting, then Linear regression
- ▶ Choose for example Model.1 as your model name in the field Enter a name for the model.
- ▶ Choose variable DISTANCE as the response variable, and variables WEIGHT and HEIGHT as the explanatory variables (keep the CTRL key pressed).
- ▶ Click on OK. The result of your linear model adjustment appears in the Output window. This result corresponds to the instructions

```
Model.1 <- lm(DISTANCE~WEIGHT+HEIGHT,data=Illness)
summary(Model.1)
```

which are shown in the Script window.

See also

Chapter 14 presents the linear model in further detail.

We now visualize the least squares plane corresponding to the fitted model.

- ▶ In the Plot menu, choose 3D plot, then 3D scatterplot....
- ▶ Choose variable DISTANCE as the response variable and the variables WEIGHT and HEIGHT as explanatory variables (use the CTRL key).
- ▶ Choose Ordinary least squares as the surface to fit. Click on OK.

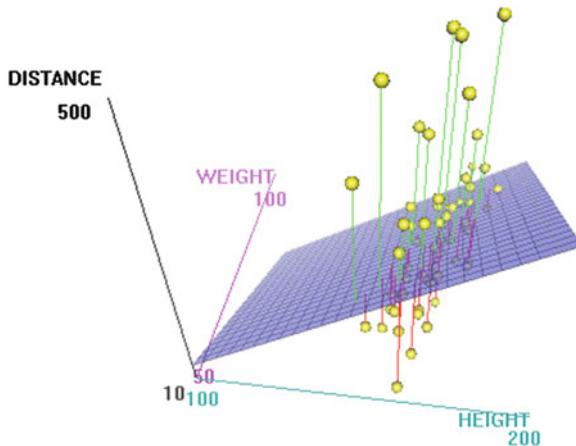


Fig. 1.8: Least squares plane

You can now see the 3D scatterplot (shown in Fig. 1.8) and the least squares plane. You can move the image with your mouse.

1.5.1.4 Adding Functionalities to the RCommander Interface

Some packages available on the official R website can also be integrated to the RCommander menus. They are easy to identify: their names start with `RcmdrPlugin`. We now illustrate how to use such a package.

See also

 You can read the article [17] which explains how to build a package for RCommander integration.

- **The TeachingDemos package**

The `RcmdrPlugin.TeachingDemos` package can be used to illustrate some statistical concepts.

- ▶ Type `install.packages("RcmdrPlugin.TeachingDemos")` in the Script window. Click on Submit and choose a nearby mirror. Once the installation is complete, close and reopen RCommander using the instruction `Commander()`.
- ▶ In the Tools menu, choose Load Rcmdr plug-ins..., click on OK and answer Yes to the question Restart now?.
- ▶ There is a new menu called Demos. In this menu, you can choose for example the submenu Simple Correlation and explore the notion of correlation.

This plug-in also adds submenus to pre-existing menus. For example, in the **Distributions** menu, you can now choose **Visualize distributions**, then **t distributions**. By checking **Show Normal Distribution**, and by playing with the **d.f. (degree of freedom)** cursor, you can visualize the closeness of the Student distribution and the normal distribution.

• The **sos** package

The **RcmdrPlugin.sos** package can be used to ease the search for help on a given concept or function. Follow the same steps as before to install this plug-in. A new submenu called **Search R Help ... (sos)** appears in the **Help** menu. Explore this new **RCommander** functionality, for example, by typing **linear model**.

See also

Chapter 6 describes how to search for information about R.



1.5.2 Using R with the Console

In the previous subsection, we saw how to use R through menus. In fact, this way of proceeding is far from optimal, since it imposes many limitations on the possibilities offered by R. Many analyses, either deeper or more recent and innovative, are not available in the **RCommander** menus. It is thus very useful to escape from the “button clicking” approach and master the R programming language. You will then be able to perform simulations and to code repetitive tasks. We have already encountered a few R instructions when using **RCommander**, which is itself a tool written in the R language. We now propose a brief introduction to a few elements of the R syntax, first through an analysis of complex data arising from a functional magnetic resonance imaging (MRI) experiment, then by letting the reader type a few R commands and think about the output.

1.5.2.1 The Strength of R Shown on an Example

Some neuroscientists work on finding which part of the brain deals with visual information on colour. To this end, a visual stimulus, consisting in an alternance of coloured and non-coloured moving patterns, is shown to a subject. During this time, volumic images of the subject’s brain are acquired at time $t = 1, \dots, T$ with an MRI scanner. Each 3D image is in fact a large (Rubik’s!) cube made of many voxels, the 3D equivalents of 2D pixels. At time $t = 1, \dots, T$, each voxel contains an electromagnetic measurement value $x(t)$. We can thus consider that in each voxel, we have observed a time series $\{x(t); t = 1, \dots, T\}$ representing

electromagnetic variations. The acquired data (given in file Mond4D.nii, produced during a Mondrian experiment performed by M. Dojat and J. Huppé) thus consist in a 4-dimensional array, the concatenation of several volumic brain images measured through time.

We used R to find, in each brain slice, which voxel had temporal variations most correlated with the stimulus signal. The code below can be downloaded from <http://biostatisticien.eu/springeR/brain-code.R> and opened, thanks to the submenu Open script... of the File menu in R. The key combination CTRL+R then executes one by one the instructions of this script. You can try to execute these instructions to visualize the results. This will help you familiarize yourself with some of the possibilities offered by R.

We first download the data files we need (the files Mondanat.img and Mondanat.hdr contain an anatomical image of the subject's brain).

```
> myfile <- function(myfile)
+ download.file(paste("http://biostatisticien.eu/springeR/",
+ myfile,sep=""),paste(getwd(),"/",myfile,sep=""),mode="wb")
> myfile("Mond4D.nii")
> myfile("Mondanat.hdr")
> myfile("Mondanat.img")
```

We then install the package to read the data.

```
> install.packages("AnalyzeFMRI") # Choose a mirror.

> # File names.
> file.func <- paste(getwd(),"/","Mond4D.nii",sep="")
> file.anat <- paste(getwd(),"/","Mondanat.img",sep="")

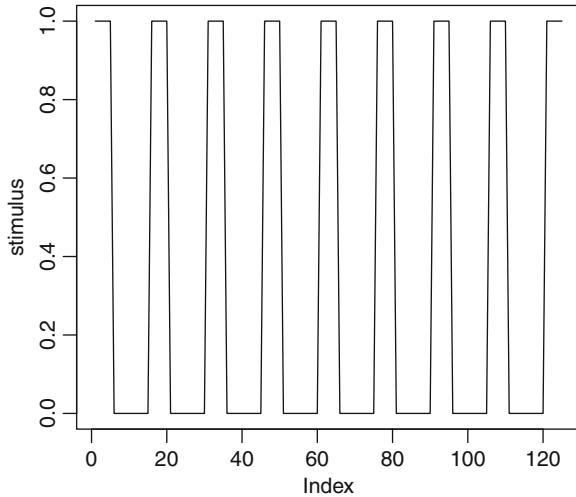
> # Brain slice number.
> slice <- 10
```

The next instructions read the data.

```
> anat.slice <- f.read.nifti.slice(file.anat,slice,1)
> class(anat.slice)
[1] "matrix"
> dim(anat.slice)
[1] 128 128
> func.slice <- f.read.nifti.slice.at.all.timepoints(file.func,
  slice)
> class(func.slice)
[1] "array"
> dim(func.slice)
[1] 128 128 125
```

We now create the coding of the visual stimulus signal (1=colour, 0=no colour).

```
> stimulus <- c(rep(c(1,1,1,1,1,0,0,0,0,0,0,0,0,0),8),1,1,1,1,1)
> plot(stimulus,type="l")
```



We compute correlations between the observed time series in each voxel and the stimulus series.

```
> corMat <- matrix(NA,nrow=128,ncol=128)
> for (i in 1:128) {
+   for (j in 1:128) {
+     corMat[i,j] <- cor(func.slice[i,j,],stimulus)
+   }
+ }
```

We can now compute the coordinates of the voxel most strongly correlated with the stimulus

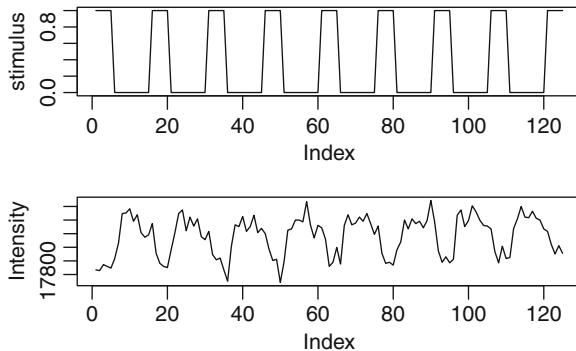
```
> which(abs(corMat)==max(abs(corMat),na.rm=TRUE),arr.ind=TRUE)
      row col
[1,] 67 117
```

and the correlation value of this voxel

```
> corMat[67,117]
[1] -0.6675017
```

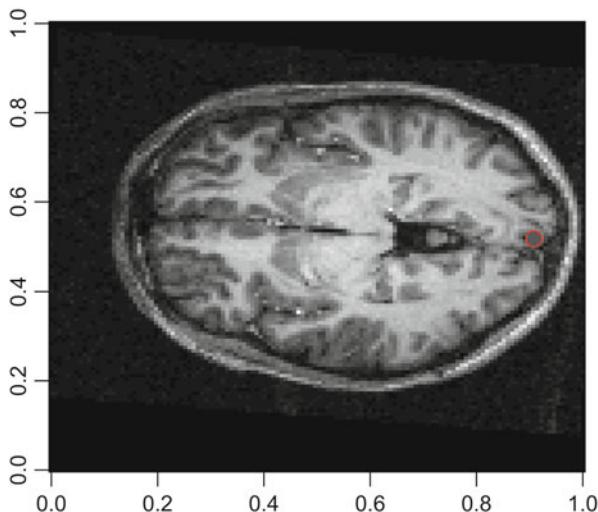
We can then plot the time series observed in this voxel.

```
> par(mfrow=c(2,1))
> plot(stimulus,type="l")
> plot(func.slice[67,117,],type="l",ylab="Intensity")
```



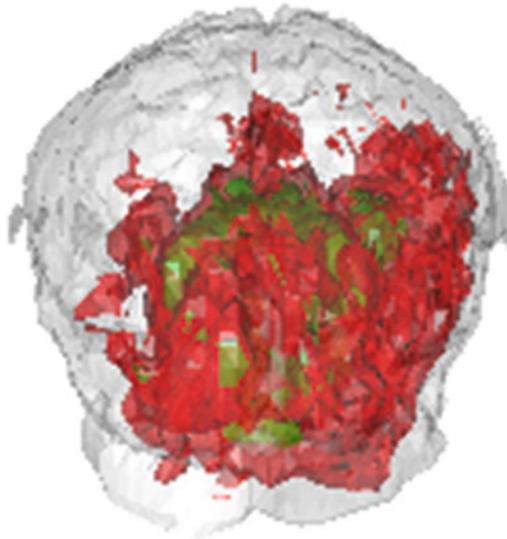
We are now able to identify on the anatomical image of the brain the most active voxel for the visual stimulus.

```
> image(as.matrix(rev(as.data.frame(t(anat.slice)))) ,  
       col=gray((0:32)/32))  
> points(117/128,67/128,col="red",cex=2,pch=19)
```



Note that you can also visualize these data in 3D. The following instructions, taken from the help file for the function `contour3d()` from package `misc3d`, give an interactive 3D view of the brain.

```
> install.packages("misc3d")  
  
> require("misc3d")  
> a <- f.read.analyze.volume(system.file("example.img",  
+                               package="AnalyzeFMRI"))  
> a <- a[,,,1]  
> contour3d(a,1:64,1:64,1.5*(1:21),lev=c(3000, 8000, 10000),  
+            alpha=c(0.2,0.5,1),color=c("white","red","green"))
```



You can try to move the image with your mouse.

1.5.2.2 A Brief Introduction of R Syntax Through Some Instructions to Type

- **Basic operations**

We advise the reader to play with these commands and try to understand how they work.

```
> 1*2*3*4
[1] 24
> factorial(4)
[1] 24
> cos(pi)
[1] -1
> x <- 1:10
> x
[1]  1  2  3  4  5  6  7  8  9 10
> exp(x)
[1]  2.718282    7.389056   20.085537   54.598150
[5] 148.413159   403.428793  1096.633158  2980.957987
[9] 8103.083928  22026.465795
> x^2
[1]  1  4  9 16 25 36 49 64 81 100
> chain <- "R is great!"
> chain
[1] "R is great!"
> nchar(chain)
[1] 11
> ?nchar
```

```
> M <- matrix(x, ncol=5, nrow=2)
> M
     [,1]   [,2]   [,3]   [,4]   [,5]
[1,]    1      3      5      7      9
[2,]    2      4      6      8     10
> M[2,3]
[1] 6
> L <- list(matrix=M, vector=x, chain=chain)
> L[[3]]
[1] "R is great!"
> while(TRUE) {
+   toguess <- sample(1:2,1)
+   {cat("Guess a number among 1, 2, 3: ") ; value <- readline()}
+   if (value == toguess) {print("Well done!") ; break()}
+   else print("Try again.")
+ }

> ls()
[1] "chain"  "L"       "M"       "x"
> rm(chain)
```

The following commands perform matrix operations:

```
> A <- matrix(runif(9), nrow=3)
> 1/A
     [,1]      [,2]      [,3]
[1,] 2.270797 1.546875 1.422103
[2,] 1.268152 1.957924 1.057803
[3,] 1.642736 5.273120 2.174020
> A * (1/A)
     [,1]   [,2]   [,3]
[1,]    1      1      1
[2,]    1      1      1
[3,]    1      1      1
> B <- matrix(1:12, nrow=3)
> A * B
Error in A * B : non-conformable arrays
> A %*% B
     [,1]      [,2]      [,3]      [,4]
[1,] 3.842855 9.212923 14.582990 19.95306
[2,] 4.646105 11.380053 18.114001 24.84795
[3,] 2.367954 6.143031 9.918107 13.69318
> (invA <- solve(A))
     [,1]      [,2]      [,3]
[1,] 1.145642 -3.376148 5.187347
[2,] 4.379786 -4.641906 2.844607
[3,] -3.321872 6.381822 -5.863772
> A %*% invA
     [,1]      [,2]   [,3]
[1,] 1.000000e+00 0.000000e+00    0
[2,] 0.000000e+00 1.000000e+00    0
[3,] -2.220446e-16 4.440892e-16    1
```

```
> det(A)
[1] 0.04857799
> eigen(A)
$values
[1] 1.6960690+0.000000i -0.1424863+0.091319i
[3] -0.1424863-0.091319i
$vectors
[,1] [,2] [,3]
[1,] 0.5859852+0i 0.6140784-0.1816841i 0.6140784+0.1816841i
[2,] 0.7064296+0i 0.2234155+0.2505528i 0.2234155-0.2505528i
[3,] 0.3969616+0i -0.6908020+0.0000000i -0.6908020+0.0000000i
```

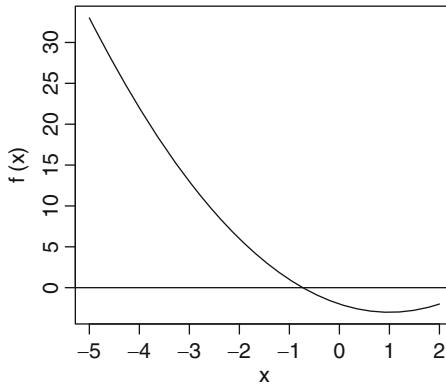
• Statistics

Here are a few statistical calculations.

```
> weight <- c(70,75,74)
> mean(weight)
[1] 73
> height <- c(182,190,184)
> mat <- cbind(weight,height)
> mat
      weight height
[1,]    70    182
[2,]    75    190
[3,]    74    184
> apply(mat,MARGIN=2,FUN=mean)
      weight height
73.0000 185.3333
> ?apply
> colMeans(mat)
      weight height
73.0000 185.3333
> names <- c("Peter","Ben","John")
> data <- data.frame(Names=names,height,weight)
> summary(data)
   Names        height        weight
Ben :1  Min.   :182.0  Min.   :70.0
John:1  1st Qu.:183.0  1st Qu.:72.0
Peter:1 Median :184.0  Median :74.0
          Mean   :185.3  Mean   :73.0
          3rd Qu.:187.0  3rd Qu.:74.5
          Max.   :190.0  Max.   :75.0
```

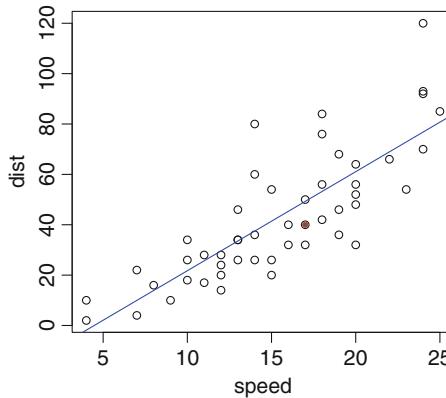
• Some plots

```
> f <- function(x) x^2-2*x-2
> curve(f,xlim=c(-5,2));abline(h=0)
> locator(1) # Click on the intersection of the two curves.
```

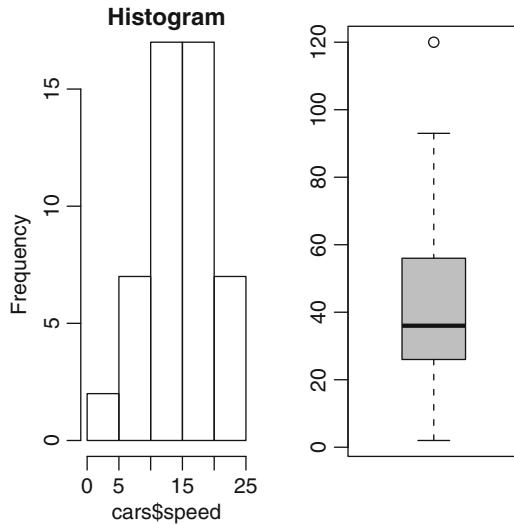


```
> uniroot(f,c(-5,2))
$root
[1] -0.7320503
$f.root
[1] -1.874450e-06
$iter
[1] 8
$estim.prec
[1] 6.103516e-05

> plot(cars)
> abline(lm(dist~speed,data=cars),col="blue")
> points(cars[30,],col="red",pch=20)
```



```
> par(mfrow=c(1,2))
> hist(cars$speed,main="Histogram")
> boxplot(cars$dist,col="orange")
```



See also

This link points to a reference card of the most useful R functions <http://cran.r-project.org/doc/contrib/Short-refcard.pdf>



Chapter 2

A Few Data Sets and Research Questions

Goals of this chapter

This chapter presents a few data sets from epidemiological studies analyzed by various teams at the Bordeaux School of Public Health (Institut de Santé publique, d’Epidémiologie et de Développement—ISPED). Each data set comes with a short research question, which will help understand the context of the study. They will be used throughout this book to show how to use the functionalities of R for importing and manipulating data and performing appropriate statistical analyses. For each data set, we give a table with a description, the variables and the coding. The reader should refer to this chapter when the data sets are mentioned later in the book. A table at the end of the chapter indicates in which chapters each data set is used. All these data sets are available online on the website associated with the book: <http://www.biostatisticien.eu/springeR>.

SECTION 2.1

Body Mass Index of Children

Presentation

A sample of 152 children (3 or 4 years old) in their first year of kindergarten in schools in Bordeaux (Gironde, SouthWest France) underwent a physical check-up in 1996–1997.

Variables and Coding

Description	Unit or coding	Variable
Gender	F for female; M for male	GENDER
School in an underprivileged area (<i>zone d'éducation prioritaire, ZEP</i>)	Y for yes; N for no	zep
Weight	Kg (to the nearest 100 g)	weight
Age at date of examination	Years	years
Age at date of examination	Months	months
Height	Cm (to the nearest 0.5 cm)	height

Data Set: BMI-CHILD

File: bmichild.xls

SECTION 2.2

Weight at Birth

Presentation

This study focused on risks associated with low weight at birth; the data were collected at the Baystate Medical Centre, Massachusetts, in 1986. Physicians have been interested in low weight at birth for several years, because underweight babies have high rates of infant mortality and infant anomalies. The behaviour of the mother-to-be during pregnancy (diet, smoking habits) can have a significant impact on the chances of having a full-term pregnancy, and thus of giving birth to a child of normal weight. The data file includes information on 189 women (identification number: ID) who came to the centre for consultation. Weight at birth is categorized as low if the child weighs less than 2,500 g.

Variables and Coding

Description	Unit or coding	Variable
Age of mother	Years	AGE
Weight of mother at last menstrual period	Pounds	LWT
Race of mother	1 = white; 2 = black; 3 = other	RACE
Smoking during pregnancy	Yes = 1; no = 0	SMOKE
Number of premature births in medical history	0 = none; 1 = one; 2 = two; etc.	PTL
Medical history of hypertension	Yes = 1; no = 0	HT
Uterine irritability	Yes = 1; no = 0	UI
Number of medical consultations during first trimester	0 = none; 1 = one; etc.	FVT
Weight at birth	Grams	BWT
Weight at birth less than 2,500 g	Yes = 1; No = 0	LOW

Data Set: WEIGHT-BIRTH

File: Birth_weight.xls

SECTION 2.3

Intima-Media Thickness**Presentation**

Atherosclerosis is the main cause of death for men above 35 and women above 45 in most developed countries. It is a thickening and hardening of internal artery walls. One of its consequences is myocardial infarction. An artery wall is made of three layers; innermost to outermost, they are called intima, media and adventitia. Intima-media thickness is a marker of atherosclerosis. It was measured by ultrasonography on a sample of 110 subjects in 1999 in Bordeaux hospitals. Information on the main risk factors was also collected.

Variables and Coding

Description	Unit or coding	Variable
Gender	1 = male; 2 = female	GENDER
Age at date of consultation	Years	AGE
Height	Cm	height
Weight	Kg	weight
	0 = non-smoker	
Smoking status	1 = former smoker	tobacco
	2 = smoker	
Estimation of tobacco consumption for smokers and former smokers	Number of packs/year	packyear
Physical activity	0 = no; 1 = yes	SPORT
Intima-media thickness	Mm	measure
	0 = non-drinker	
Alcohol consumption	1 = occasional drinker	alcool
	2 = regular drinker	

Data Set: INTIMA-MEDIA**File:** Intima_Media_Thickness.xls

SECTION 2.4

Diet of Elderly People

Presentation

A sample of 226 elderly people living in Bordeaux (Gironde, South-West France) were interviewed in 2000 for a nutritional study.

Variables and Coding

Description	Unit or coding	Variable
Gender	2 = female; 1 = male 1 = single	gender
Family status	2 = living with spouse 3 = living with family 4 = living with someone else	situation
Daily consumption of tea	Number of cups	tea
Daily consumption of coffee	Number of cups	coffee
Height	Cm	height
Weight	Kg	weight
Age at date of interview	Years 0 = never 1 = less than once a week 2 = Once a week 3 = 2/3 times a week 4 = 4/6 times a week 5 = every day	age
Consumption of meat	Idem	meat
Consumption of fish	Idem	fish
Consumption of raw fruits	Idem	raw_fruits
Consumption of cooked fruits and vegetables	Idem	cooked_fruits_veg
Consumption of chocolate	Idem 1 = butter 2 = margarine 3 = peanut oil	chocol
Type of fat used for cooking	4 = sunflower oil 5 = olive oil 6 = mix of vegetable oils (e.g., Isio4) 7 = colza oil 8 = duck or goose fat	fat

Data Set: NUTRIELDERLY

File: nutrition_elderly.xls

SECTION 2.5

Study Case of Myocardial Infarction

Presentation

The study for which the following data were collected aimed at examining whether women who use or have used oral contraceptives are at a higher risk of myocardial infarction. The sample includes 149 women who had myocardial infarction (cases) and 300 women who did not (controls). The main exposure factor is usage of oral contraceptives; the data also include age, weight, height, tobacco consumption, hypertension and family history of cardiovascular diseases.

Variables and Coding

Description	Unit or coding	Variable
Myocardial infarction	0 = controls; 1 = cases	infarct
Usage of oral contraceptives	0 = never; 1 = yes	co
	0 = no	
Tobacco usage	1 = smoker	tobacco
	2 = former smoker	
Age	Years	age
Weight	Kg	weight
Height	Cm	height
Family history of cardiovascular diseases	0 = no; 1 = yes	atcd
Hypertension	0 = no; 1 = yes	hta

Data set: INFARCTION

File: Infarction.xls

SECTION 2.6

Summary Table of Use of Data Sets

	Import– export	Manipulation	Methods			
			Descriptive statistics	Tests	ANOVA	Regression
BMI-	×	×		×		
CHILD						
WEIGHT-	×	×				
BIRTH						
INTIMA-	×	×		×	×	×
MEDIA						
NUTRI	×	×	×	×		
ELDERLY						
INFARCTION	×		×			

Part II
The Bases of R

Chapter 3

Basic Concepts and Data Organisation

Goals of this chapter

This chapter introduces the basic concepts of the R software (calculator mode, assignment operator, variables, functions, arguments) and the various data types and structures which can be handled by R.

— SECTION 3.1 —

Your First Session

Launch R by double-clicking its icon on the Windows Desktop (or from the Start menu). At the end of the text displayed in the *R console*, you can see the **prompt symbol** >, inviting you to type in your first instruction in the R language.

```
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
```

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

```
>
```

For example, type "R is my friend", then validate by hitting the ENTER key (or RETURN). You will then get

```
> "R is my friend"
[1] "R is my friend"
```

As you can see, R is well behaved and kindly proceeds with your request. This will usually be the case—maybe R is trying to compensate for its lack of conviviality. We shall explain later on why R's reply starts with [1].

3.1.1 R Is a Calculator

Like many other similar languages, R can easily replace all the functionalities of a (very sophisticated!) calculator. One of its major strengths is that it also allows calculations on arrays. Here are a few very basic examples.

```
> 5*(-3.2)           # Careful: the decimal mark must be a point (.)
[1] -16
> 5*(-3,2)          # otherwise, the following error is generated:
Error : ',' unexpected in "5*(-3,"

> 5^2                # Same as 5**2.
[1] 25
> sin(2*pi/3)
[1] 0.8660254
> sqrt(4)            # Square root of 4.
[1] 2
> log(1)              # Natural logarithm of 1.
[1] 0
> c(1,2,3,4,5)        # Creates a collection of the first five
# integers.
[1] 1 2 3 4 5
> c(1,2,3,4,5)*2    # Calculates the first five even numbers.
[1] 2 4 6 8 10
```

Tip

Any R code after the symbol "#" is considered by R as a comment. In fact, R does not interpret it.



You can now exit the R software by typing the following instruction: q().

You are asked whether you wish to save an image of the session. If you answer yes, the commands you typed earlier will be accessible again next time you open R, by using the "up" and "down" keyboard arrows.

3.1.2 Displaying Results and Variable Redirecting

As you have probably noticed, R responds to your requests by displaying the result obtained after evaluation. **This result is displayed, then lost.** At first, this might seem sensible, but for more advanced uses, it is useful to redirect the R output to your request, by storing it in a variable: this operation is called **assigning the result to a variable**. Thus, an assignment evaluates an expression but does not display the result, which is instead stored in an object. To display the result, all you need to do is type the name of that object, then hit ENTER.

To make an assignment, use the **assignment arrow** `<-`. To type the arrow `<-`, use the lesser than symbol (`<`) followed by the minus symbol (`-`).

To create an object in R, the syntax is thus

```
Name.of.the.object.to.create <- instructions
```

For example,

```
> x <- 1      # Assignment.  
> x          # Display.  
[1] 1
```

We say that the value of `x` is 1, or that we have assigned 1 to `x` or that we have stored in `x` the value 1. Note that the assignment operation can also be used the other way around `->`, as in

```
> 2 -> x  
> x  
[1] 2
```

Warning

The symbol `=` can also be used, but its use is less general and is therefore not advised. Indeed, mathematical equality is a symmetrical relation with a specific meaning, very different to assignment. Furthermore, there are cases where using the symbol `=` does not work at all.



Tip

Note that a pair of brackets allows you to assign a value to a variable and display the evaluation result at the same time:

```
> (x <- 2+3)  
[1] 5
```



If a command is not complete at the end of a line, R will display a different prompt symbol, by default the plus sign (+), on the second line and on following lines. R will continue to wait for instructions until the command is syntactically complete.

```
> 2*8*10+exp(1)
[1] 162.7183
> 2*8*
+ 10+exp(
+ 1)
[1] 162.7183
```

Warning

Here are the **rules for choosing a variable name** in R: a variable name can only include alphanumerical characters as well as the dot (.); variable names are *case sensitive*, which means that R distinguishes upper and lower case; a variable name may not include white space or start with a digit, unless it is enclosed in quotation marks "".



3.1.3 Work Strategy

- Take the habit of storing your files in a folder reserved to this effect (you could call it **Rwork**). We also advise you to type all your R commands in a script window called *script* or *R editor*, accessible through the “File/New script” menu. Open a new script window, click on the “Windows/Side by side” menu, then copy the script below:

```
x <- 5*(-3.2)
5^2
sin(2*pi/3)
sqrt(4)
c(1,2,3,4,5)
z <- c(1,2,3,4,5)*2
```

Mac



On a Mac, the menu is “File/New Document”, and it is not possible to lay the windows side by side.

At the end of your session, you can save this script in the folder **Rwork**, for example, as **myscript.R**, and reopen it during a later session from the menu “File/Open a script” (or on a Mac “File/Open Document”).

- You can then use the key combinations **CTRL+A** (**COMMAND+A** on a Mac) to select all the instructions, then **CTRL+R** (**COMMAND+ENTER** on a Mac) to paste and execute them in one step in the R console. You can also execute a single line of R instructions from the script by hitting **CTRL+R** when the blinking cursor is on the relevant line of the script window.

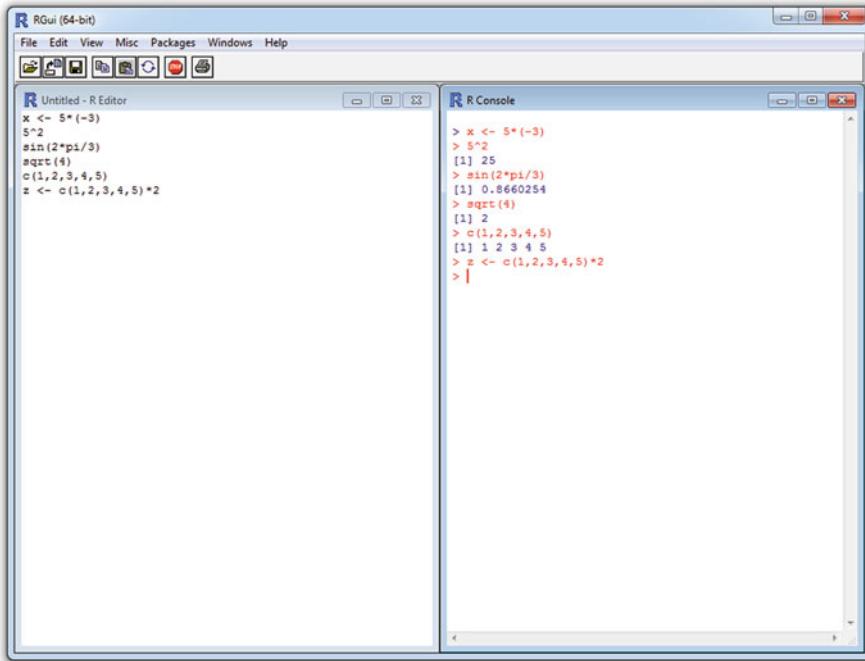


Fig. 3.1: The script window and the command console

Tip

Note in Fig. 3.1 the presence of the red STOP button that lets you interrupt a calculation that would last too long.



You can also use the function `source()` from the R console to read and execute the content of your file. This helps prevent overloading the console, as we will see later. You may find it useful to proceed as follows:

- Clicking once in the *R console* window.
- Going to the menu “File/Change current directory” (“Misc/Change work directory” on a Mac).
- Exploring your file system and selecting the folder `Rwork`.
- Typing in the console `source("myscript.R")`. Note that for the above example, the use of this instruction will not produce any output. The following Do it yourself will clarify this point.

Do it yourself

Begin to create a folder called **Rwork** in your home directory. Then, type in and save in an **R** script the preceding instructions. The file containing the **R** script will be called **myscript.R** and will be put in **Rwork**. Now close then reopen **R**. Next, type the following instructions in the **R** console:

```
rm(list=ls()) # Delete all existing objects.  
ls()           # List existing objects.  
source("myscript.R")  
ls()  
x  
z
```

Note that the **source()** function has permitted to execute the preceding instructions. You may have noticed that the computations which have not been redirected into variables have not been printed. So their result is lost. Change your script and add the following instructions at the end of it:

```
print(2*3)  
print(x)
```

Save it, then source it. What happened?

- Take the habit of using the online **R** help. The help is very complete and in English. You can reach it with the function **help()**. For example, type **help(source)** to get help about the function **source()**.

See also

All these notions will be examined in further detail in Chaps. 6 and 9.

Tip

Two good code editors are RStudio, available at <http://www.rstudio.com>, and Tinn-R (Windows only), available at <http://www.sciviews.org/Tinn-R>. They offer a better interaction between a script's code and its execution. They also provide syntactic colouring of the code.

**Linux**

Under Linux, note that the editors JGR and Emacs/ESS are available.



See also

You can consult the list of R editors on the webpage http://www.sciviews.org/_rgui/projects/Editors.html.

**Do it yourself**

The body mass index (BMI) is used to determine a person's corpulence. It is calculated using the formula

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height}^2 (\text{m})}.$$

Calculate your BMI. You simply need to type the following lines in your script window:

```
# You can type 2 instructions
# on the same line thanks to the symbol ;
My.Weight <- 75 ; My.Height <- 1.90
My.BMI <- My.Weight/My.Height^2
My.BMI
```

Execute this script by using the work strategy mentioned earlier. You can then modify this script to calculate your own BMI.

We propose a function to visualize your corpulence type. Execute the following instructions:

```
source("http://www.biostatisticien.eu/springeR/BMI.R",
encoding="utf8")
display.BMI(My.BMI)
```

You will learn how to program this kind of result in later chapters.

3.1.4 Using Functions

We have already encountered a few functions: `sin()`, `sqrt()`, `exp()` and `log()`. The base version of R includes many other functions, and thousands of others can be added (by installing packages or by creating them from scratch).

Note that a function in R is defined by its **name** and by the list of its **parameters**. Most functions output a **value**, which can be a number, a vector, or a matrix.

Using a function (or **calling** or **executing** it) is done by typing its name followed, in brackets, by the list of (formal) arguments to be used. Arguments are separated by commas. Each argument can be followed by the sign = and the value to be given to the argument. This value of the formal argument will be called effective argument, call argument or sometimes entry argument.

We will therefore use the instruction

```
functionname(arg1=value1,arg2=value2,arg3=value3)
```

where arg1, arg2, ... are called the arguments of the function, whereas value1 is the value given to the argument arg1, etc. Note that you do not necessarily need to indicate the names of the arguments, but only the values, as long as you follow their order.

For any R function, some arguments must be specified and others are optional (because a default value is already given in the code of the function).

Warning

Do not forget the brackets when you call a function. A common mistake for beginners is forgetting the brackets:

```
> factorial
function (x)
gamma(x + 1)
<environment: namespace:base>
> factorial(6)
[1] 720
```



The output to the first instruction gives the code (i.e. the recipe) of the function, whereas the second instruction executes that code. This is also true for functions which do not require an argument, as shown in the following example:

```
> date()
[1] "Wed Jan 9 16:04:32 2013"
> date
function ()
.Internal(date())
<environment: namespace:base>
```

Obviously, this is not the place to comment the code of these functions.

To better understand how to use arguments, take the example of the function `log(x, base=exp(1))`. It can take two arguments: `x` and `base`.

The argument `x` must be specified: it is the number of which we wish to calculate the logarithm. The argument `base` is optional, since it is followed with the symbol = and the default value `exp(1)`.

Tip

An argument which is not followed with the symbol = must be specified. A parameter is optional if it is followed with =.



In the following code, R will calculate the *natural* logarithm of the number 1, since the base argument is not specified:

```
> log(1)
[1] 0
```

Note

For some functions, no argument needs to be specified, for example, `matrix`, which we shall encounter later on.



One last important note is that **you can call a function by playing with the arguments in several different ways**. This is an important feature of R which makes it easier to use, and you will find it useful to understand this principle. To calculate the natural logarithm of 3, any of the following expressions can be used:

<code>log(3)</code>	<code>log(3,base=exp(1))</code>
<code>log(x=3)</code>	<code>log(3,exp(1))</code>
<code>log(x=3,base=exp(1))</code>	<code>log(base=exp(1),3)</code>
<code>log(x=3,exp(1))</code>	<code>log(base=exp(1),x=3)</code>

Warning

Note that calling

```
log(exp(1), 3)
```

will calculate the logarithm of `exp(1)` in base 3.



Finally, recall that we have been able to see the code for the function `factorial()`:

```
> factorial
function (x)
gamma(x + 1)
<environment: namespace:base>
```

This function was defined by the R developers with the following instructions:

```
> factorial <- function(x) gamma(x+1)
```

It is very easy to code a new function in R, by using the function `function()`. For example, here is how to code a function which takes two arguments n and p and calculates the binomial coefficient $\binom{n}{p} = \frac{n!}{p!(n-p)!}$:

```
> binomial <- function(n,p) factorial(n)/(factorial(p)*
+                               factorial(n-p))
```

You can then use this new function as any other R function:

```
> binomial(4,3)
[1] 4
```

We shall study in much further detail how to create more elaborate functions in Sect. 5.8 and in Chap. 8.

Note



In fact, there already exists an R function to compute the Newton binomial coefficient. This is the function `choose()` that works more efficiently, especially for big numbers.

SECTION 3.2

Data in R

R, like most computer languages, can handle classical data types. R is actually able to automatically recognize data types according to the format of the input. One of the main strengths of R is its ability to organize data in a structured way. **This will turn out to be very useful for many statistical procedures we will study later on.**

3.2.1 Data Nature (or Type, or Mode)

Data “types” can be handled using the functions `mode()` and `typeof()`, which only differ in very subtle ways which we shall ignore.

Note



The function `class()` is more general: it is used to handle both data type and structuring. We shall study it later on. For ease of understanding, we shall use the command `typeof()`.

The various types (or modes) of data are now presented.

3.2.1.1 Numeric Type (`numeric`)

There are two numeric types: integers (`integer`) and real numbers (`double`).

If you enter

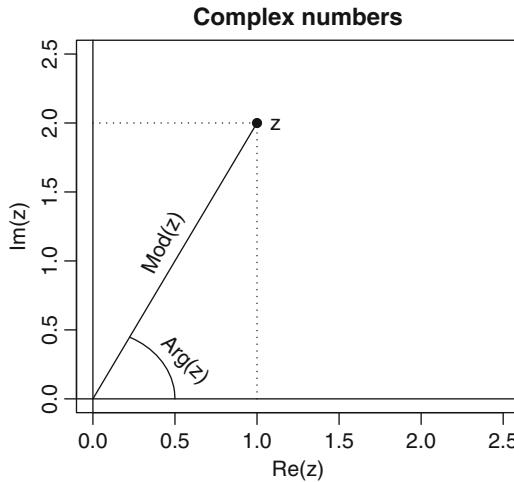


Fig. 3.2: Characteristics of a complex number

```
> a <- 1
> b <- 3.4
> c <- as.integer(a)
> typeof(c)
[1] "integer"
```

the variables `a` and `b` are of the type "double", and the variable `c` has the same value as `a`, except that it has been forced to be of the type "integer". This is useful because a vector of "integer"s takes up less memory space than a vector of "double"s of the same length. Instructions starting with `as.` are very common in R to convert data into a different type. We will see in the Sect. 3.2.2.1 how to check that an object's type is numeric.

3.2.1.2 † Complex Type (`complex`)

A complex number is created, thanks to the letter `i`. The functions `Re()` for real part, `Im()` for imaginary part, `Mod()` for modulus and `Arg()` for argument can be used (Fig. 3.2).

Here are a few examples:

```
> 1i
[1] 0+1i
> z <- 1+2i
> typeof(z)
[1] "complex"
> is.complex(z) # To know whether an object is of the complex
# type.
[1] TRUE
> Re(z)
[1] 1
```

```
> Im(z)
[1] 2
> Mod(z)
[1] 2.236068
> Arg(z)
[1] 1.107149
```

3.2.1.3 Boolean or Logical Type (logical)

The type `logical()` is the result of a logical operation. It can take the values `TRUE` or `FALSE`. Here are a few instructions to create logical values:

```
> b>a
[1] TRUE
> a==b
[1] FALSE
> is.numeric(a)
[1] TRUE
> is.integer(a)
[1] FALSE
> x <- TRUE
> is.logical(x)
[1] TRUE
```

Warning

 `TRUE` and `FALSE` can also be entered in a more condensed form by typing `T` and `F`, respectively. But this should not be encouraged.

When needed, this data type is naturally converted to numeric without having to specify the conversion: `TRUE` is worth `1` and `FALSE` is worth `0`. The following example illustrates this point:

```
> TRUE + T + FALSE*F + T*FALSE + F
[1] 2
```

3.2.1.4 Missing Data (NA)

A missing or undefined value is indicated by the instruction `NA` (for *non-available*). Several functions exist to handle this data type. In fact, R considers this data type as a constant logical value. Strictly speaking, it is therefore not a data type. Here are a few examples which use the instruction `NA`:

```
> x <- c(3,NA,6)
> is.na(x)
[1] FALSE  TRUE FALSE
> mean(x)          # Trying to calculate the mean of x.
```

```
[1] NA
> mean(x,na.rm=TRUE)      # The na.rm argument means that NA's
                           # should be ignored (NA.remove).
[1] 4.5
```

This is a very important notion when it comes to reading statistical data files. We shall examine it in further detail in Chap. 5.

Warning

Do not mistake NA for the reserved word NaN, which means *not a number*:

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

Note also that the following instruction does not output NaN but infinity, represented in R with the reserved word Inf.

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



3.2.1.5 Character String Type (`character`)

Any information between quotation marks (single ' or double ") corresponds to a character string:

```
> a <- "R is my friend"
> mode(a)
[1] "character"
> is.character(a)
[1] TRUE
```

Conversions into a character string from another type are possible. Converting a character string into another type is possible as long as R can correctly interpret the content inside the quotations marks. Note that some conversions are done automatically. Here are a few examples:

```
> as.character(2.3)           # Conversion into a character string.
[1] "2.3"
> b <- "2.3"
> as.numeric(b)             # Conversion from a character string.
[1] 2.3
> as.integer("3.4")         # Conversion from a character string.
[1] 3
> c(2,"3")
[1] "2" "3"
> as.integer("3.four")      # Impossible conversion.
[1] NA
```

Note

The differences between single and double quotation marks are given in Chap. 5.

3.2.1.6 † Raw Data (`raw`)

In R, it is possible to work directly with bytes (displayed in hexadecimal format). This can sometimes be useful when reading certain files in binary format. We shall see examples in Chap. 7.

```
> x <- as.raw(15)
> x
[1] 0f
> mode(x)
[1] "raw"
```

Summary

Table 3.1: The various data types in R

Data type	Type in R	Display
Real number (integer or not)	<code>numeric</code>	3.27
Complex number	<code>complex</code>	3+2i
Logical (true/false)	<code>logical()</code>	TRUE or FALSE
Missing	<code>logical()</code>	NA
Text (string)	<code>character</code>	"text"
Binary	<code>raw</code>	1c

Tip

The function `storage.mode()` get or set the type or storage mode of an object.

3.2.2 *Data Structures*

In R, you can organize (structure) the various data types defined above (Table 3.1). The structures we are about to present can be accessed or created with the function `class()` (Table 3.2).

3.2.2.1 Vectors (vector)

This is the simplest data structure. It represents a **sequence of data points of the same type**. A vector can be created with the function `c()` (for collection or concatenation). Other functions such as `seq()` or a colon `:` can also be used to create a vector. Note that when creating a vector, it is possible to mix data of different types. R will then make an implicit conversion into the more general data type, as shown in the following example:

```
> c(3, 1, 7)
[1] 3 1 7
> c(3, TRUE, 7)
[1] 3 1 7
> c(3,T,"7")
[1] "3"    "TRUE"  "7"
> seq(from=0,to=1,by=0.1)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(from=0,to=20,length=5)
[1] 0 5 10 15 20
> vec <- 2:36
> vec
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
[20] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

Warning

The indications `[1]` and `[20]` give the rank in the vector `vec` of the element they precede.



Note that it is possible to “name” the elements of a vector using the function `names()`.

```
> vec <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
> names(vec) <- letters[1:9] # 9 first letters of the alphabet.
> vec
a b c d e f g h i
1 3 6 2 7 4 8 1 0

> is.vector(vec)
[1] TRUE
> x <- 1:3
> x
[1] 1 2 3
> y <- c(1,2,3)
> y
[1] 1 2 3
> class(x)
[1] "integer"
> class(y)
[1] "numeric"
```

One would actually expect to see appear "vector of doubles" or "vector of integers" instead of "numeric" or "integer", but no software is perfect!

Advanced users



Note that the instructions `c()` and `:` give the same output, but that `x` and `y` are stored internally in different ways. The type `integer` uses less memory than the type `numeric`.

3.2.2.2 Matrices (`matrix`) and Arrays (`array`)

These two notions are generalizations of the vector notion: they represent sequences with two indices for matrices and with multiple indices for arrays. As with vectors, **elements must be of the same type, otherwise implicit conversions will occur.**

The following instruction

```
> X <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE)
> X
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

creates (and stores in the variable `X`) a matrix with four rows and three columns, filled by row (`byrow =TRUE`) with the elements of the vector `1:12` (e.g., the twelve first integers).

Similarly, a matrix can be filled by column (`byrow=FALSE`).

```
> Y <- matrix(1:12,nrow=4,ncol=3,byrow=FALSE)
> Y
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> class(Y)
[1] "matrix"
```

The function `array()` is used to create multidimensional matrices with more than two dimensions, as shown in the following figure (for a three-dimensional array) (Fig. 3.3):

```
> X <- array(1:12,dim=c(2,2,3))
> X
, , 1
     [,1] [,2]
[1,]    1    3
```

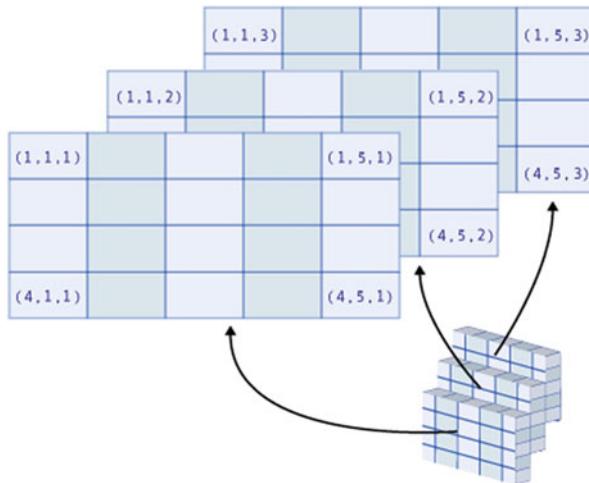


Fig. 3.3: Illustration of an array

```
[2,]     2     4
, , 2
[1,1] [1,2]
[1,]     5     7
[2,]     6     8
, , 3
[1,1] [1,2]
[1,]     9    11
[2,]    10    12
> class(X)
[1] "array"
```

Warning

Arrays with more than three dimensions can be created, thanks to the argument `dim`, which can be of length greater than 3.



3.2.2.3 Lists (list)

The most flexible and richest structure in R is the list. Unlike the previous structures, lists can **group together in one structure data of different types** without altering them. Generally speaking, each element of a list can thus be a vector, a matrix, an array or even a list. Here is a first example:

```
> A <- list(TRUE,-1:3,matrix(1:4,nrow=2),c(1+2i,3),
+           "A character string")
> A
```

```

[[1]]
[1] TRUE
[[2]]
[1] -1  0  1  2  3
[[3]]
 [,1] [,2]
[1,]    1    3
[2,]    2    4
[[4]]
[1] 1+2i 3+0i
[[5]]
[1] "A character string"
> class(A)
[1] "list"

```

In such a structure, with heterogeneous data types, element ordering is often completely arbitrary. Elements can therefore be explicitly named, which makes the output more user-friendly. Here is an example:

```

> B <- list(my.matrix=matrix(1:4,nrow=2),
+             my.complex.numbers=c(1+2i,3))
> B
$my.matrix
 [,1] [,2]
[1,]    1    3
[2,]    2    4
$my.complex.numbers
[1] 1+2i 3+0i
> list1 <- list(my.complex.number=1+1i,my.logical.value=FALSE)
> list2 <- list(my.string="I am learning R",my.vector=1:2)
> C <- list("My first list"=list1,My.second.list=list2)
> C
$'My first list'
$'My first list'$my.complex.number
[1] 1+1i
$'My first list'$my.logical.value
[1] FALSE
$My.second.list
$My.second.list$my.string
[1] "I am learning R"
$My.second.list$my.vector
[1] 1 2

```

See also

 Naming elements will make it easier to extract elements from a list (see Chap. 5, p. 106).

3.2.2.4 The Individual×Variable Table (`data.frame`)

The individual×variable table is the quintessential structure in statistics. In R, this notion is expressed by a `data.frame`. Conceptually speaking, it is a matrix with each line corresponding to an individual and each column corresponding to a variable measured on the individuals. **Each column represents a single variable, which must be of the same type across all individuals.** The columns of the data matrix can have names. Here is an example of a `data.frame` creation:

```
> BMI <- data.frame(Gender=c("M", "F", "M", "F", "M", "F"),
+   Height=c(1.83, 1.76, 1.82, 1.60, 1.90, 1.66),
+   Weight=c(67, 58, 66, 48, 75, 55),
+   row.names=c("Jack", "Julia", "Henry", "Emma", "William", "Elsa"))
> BMI
   Gender Height Weight
Jack      M    1.83    67
Julia     F    1.76    58
Henry     M    1.82    66
Emma      F    1.60    48
William   M    1.90    75
Elsa      F    1.66    55
> is.data.frame(BMI)
[1] TRUE
> class(BMI)
[1] "data.frame"
> str(BMI)
'data.frame':       6 obs. of  3 variables:
 $ Gender: Factor w/ 2 levels "F", "M": 2 1 2 1 2 1
 $ Height: num  1.83 1.76 1.82 1.6 1.9 1.66
 $ Weight: num  67 58 66 48 75 55
```

Note

The `str()` function enables one to display the structure of each column of a `data.frame`.



Advanced users

A `data.frame` can be seen as a list of vectors of identical length. This is actually how R stores a `data.frame` internally.



```
> is.list(BMI)
[1] TRUE
```

3.2.2.5 Factors (`factor`) and Ordinal Variables (`ordered`)

In R, character strings can be organized in a more astute way, thanks to the function `factor()`:

```
> x <- factor(c("blue", "green", "blue", "red",
+                           "blue", "green", "green"))
> x
[1] blue green blue red   blue green green
Levels: blue green red
> levels(x)
[1] "blue"  "green" "red"
> class(x)
[1] "factor"
```

Tip

The function `cut()` enables one to recode a continuous variable into a factor.



```
> Poids <- c(55,63,83,57,75,90,73,67,58,84,87,79,48,52)
> cut(Poids,3)
[1] (48,62] (62,76] (76,90] (48,62] (62,76] (76,90] (62,76]
[8] (62,76] (48,62] (76,90] (76,90] (76,90] (48,62] (48,62]
Levels: (48,62] (62,76] (76,90]
```

Factors can of course be used in a `data.frame`.

R indicates the different *levels* of the factor. The function `factor()` should thus be used to store qualitative variables. For ordinal variables, the function `ordered()` is better suited:

```
> z <- ordered(c("Small", "Tall", "Average", "Tall", "Average",
+                 "Small", "Small"), levels=c("Small", "Average", "Tall"))
> class(z)
[1] "ordered" "factor"
```

The `levels` argument of the function `ordered` is used to specify the order of the variable's modalities.

See also



Examples of uses of these two functions are given in Chap. 11, pp. 341 and 342.

Tip

The function `gl()` generates factors by specifying the pattern of their levels:

```
> gl(n = 2, k = 8, labels = c("Control", "Treat"))
[1] Control Control Control Control Control Control Control Control
[8] Control Treat   Treat   Treat   Treat   Treat   Treat
[15] Treat   Treat
Levels: Control Treat
```



In the above instruction, `n` and `k` are two integers, the first one giving the number of levels and the second one the number of replications.

Advanced users

A vector of character strings can be organized in a more efficient way by taking into account repeated elements. This approach allows better management of the memory: each element of the factor or of the ordinal variable is in fact coded as an integer.



3.2.2.6 Dates

R can be used to structure the data representing dates, using the `as.Date()` function for example.

```
> dates <- c("92/27/02", "92/02/27", "92/01/14",
+           "92/02/28", "92/02/01")
> dates <- as.Date(dates, "%y/%m/%d")
> dates
[1] NA          "1992-02-27" "1992-01-14" "1992-02-28"
[5] "1992-02-01"
> class(dates)
[1] "Date"
```

We will return in detail on the functions for manipulating dates in Chap. 5.

3.2.2.7 Time Series

When data values are indexed by time, it may be useful, using the `ts()` function, to organize them into an R structure that reflects the temporal aspect of these data.

```
> ts(1:10, frequency = 4, start = c(1959, 2)) # 2nd Quarter of
                                                # 1959.
      Qtr1 Qtr2 Qtr3 Qtr4
1959         1    2    3
1960         4    5    6    7
1961         8    9   10
```

See also

The reader may consult with profit the book [40] which outlines the basic techniques for modelling time series, present the R functions to use for these models and give applications of these functions on several real data sets.

Summary

Table 3.2: The various data structures in R

Data structure	Instruction in R	Description
Vector	<code>c()</code>	Sequence of elements of the same nature
Matrix	<code>matrix()</code>	Two-dimensional table of elements of the same nature
Multidimensional table	<code>array()</code>	More general than a matrix; table with several dimensions
List	<code>list()</code>	Sequence of R structures of any (and possibly different) nature
Individual×variable table	<code>data.frame()</code>	Two-dimensional table where a row represents an individual and a column represents a variable (numerical or factor). The columns can be of different natures, but must have the same length
Factor	<code>factor()</code> , <code>ordered()</code>	Vector of character strings associated with a modality table
Dates	<code>as.Date()</code>	Vector of dates
Time series	<code>ts()</code>	Time series, containing the values of a variable observed at several time points

Memorandum

<- , ->: variable assignment arrows
`mode()`, `typeof()`: gives the nature of an object
`is.numeric()`: determine whether an object is numerical
`TRUE`, `FALSE`, `is.logical()`: True, False, determine whether an object is a Boolean
`is.character()`: determine whether an object is a character string
`NA`, `is.na()`: missing value, determine whether a value is missing
`class()`: determine the structure of an object
`c()`: create a sequence of elements of the same nature
`matrix()`, `array()`: create a matrix, a multidimensional table
`list()`: create a list (collection of different structures)
`data.frame()`: create an individual×variable table
`factor()`: create a factor



Exercises

- 3.1-** What is the output of this instruction: `1:3^2` ?
- 3.2-** What is the output of this instruction: `(1:5)*2` ?
- 3.3-** What is the output of these instructions: `var<-3? Var*2?`
- 3.4-** What is the output of these instructions: `x<-2? 2x<-2*x?`
- 3.5-** What is the output of these instructions: `root.of.four <- sqrt(4)? root.of.four?`
- 3.6-** What is the output of these instructions: `x<-1? x< -1?`
- 3.7-** What is the output of this instruction: `An even number <- 16?`
- 3.8-** What is the output of this instruction: `"An even number" <- 16?`
- 3.9-** What is the output of this instruction: `"2x" <- 14?`
- 3.10-** What is the output of this instruction: `An even number?`
- 3.11-** Two symbols have been removed from this R output. What are they?

```
> 2
+
[1] 6
```

- 3.12-** What is the output of this instruction: `TRUE + T +FALSE*F + T*FALSE +F?`
- 3.13-** Name the five data types in R.
- 3.14-** Give the R instruction which gives the following output:

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

- 3.15-** Name the data structures (classes) available in R.



Worksheet

Study of Body Mass Index

We wish to analyze the characteristics of a sample of children. These children went through a medical examination in their first year of kindergarten in 1996–1997 in schools in Bordeaux (South West France). The sample below contains information on ten children between the ages of 3 and 4.

The following information is available for each child:

- gender: G for girls and B for boys;
- whether their school is in a ZEP (*zone d'éducation prioritaire*: area targeted for special help in education, recognized as socially deprived): Y for yes and N for no;
- age in years and months (two variables: one for years and one for months);
- weight in kg, rounded to the nearest 100 g;
- Height in cm, rounded to the nearest 0.5 cm.

Name	Edward	Cynthia	Eugene	Elizabeth	Patrick	John	Albert	Lawrence	Joseph	Leo
Gender	G	G	B	G	B	B	B	B	B	B
ZEP	Y	Y	Y	Y	N	Y	N	Y	Y	Y
Weight	16	14	13.5	15.4	16.5	16	17	14.8	17	16.7
Years	3	3	3	4	3	4	3	3	4	3
Months	5	10	5	0	8	0	11	9	1	3
Height	100.0	97.0	95.5	101.0	100.0	98.5	103.0	98.0	101.5	100.0

In statistics, it is of the utmost importance to know the type of the variables under study: qualitative, ordinal or quantitative. These types can be specified in R, thanks to the structure functions we introduced earlier in this chapter.

Try the following manipulations under R. Remember to use the work strategy we presented at the beginning of the chapter.

- 3.1- Choose the best R function to save the data from each variable in vectors which you will call `Individuals`, `Weight`, `Height` and `Gender`.
- 3.2- Where possible, calculate the mean of the variables.
- 3.3- Calculate the BMI of the individuals. Group the results in a vector called `BMI` (be careful of the units).

- 3.4-** Group these variables in the R structure which seems most appropriate.
- 3.5-** Use R's online help to get information on the `plot()` function.
- 3.6-** Make a scatter plot of Weight as a function of Height. Remember to add a title to your graph and to label your axes.

Chapter 4

Importing, Exporting and Producing Data

Prerequisites and goals of this chapter

- Chapter 3.
- This chapter describes the instructions to enter data in R. It presents the various possibilities R offers to import or export data, to and from software as different as Excel, SPSS, Minitab, SAS or Matlab. It also shows how to interact with databases (SQL queries). You may benefit from reading the (very complete) manual <http://cran.r-project.org/doc/manuals/R-data.pdf>.

— SECTION 4.1 —

Importing Data

4.1.1 Importing Data from an ASCII Text File

Either your data are already available in a text file in the ASCII format or you can enter them by hand using a text editor such as Wordpad under Microsoft Windows or Emacs under Linux.

Note

Entering data by hand can be done for a small number of values. If you are dealing with large amounts of data, it is more convenient to use a spreadsheet (see the next section).



The three main R functions to import data from a text file are presented in the following table (Table 4.1).

Table 4.1: Data importation functions

Function name	Description
<code>read.table()</code>	Best suited for data sets presented as tables, as it is often the case in statistics
<code>read.ftable()</code>	Reads contingency tables
<code>scan()</code>	Much more flexible and powerful. Use this in all other cases

4.1.1.1 Reading Data with `read.table()`

The following R instruction will read the data present in a file (to be chosen in a dialogue window) and import them into R as a data.frame which we have chosen to call `my.data`.

```
my.data <- read.table(file=file.choose(), header=TRUE, sep="\t",
                      dec=". ", row.names=1)
```

The function `read.table()` accepts many arguments; the most common are described in the following table (Table 4.2).

Table 4.2: Main arguments to `read.table()`

Argument name	Description
<code>file=path/to/file</code>	Location and name of the file to be read
<code>header=TRUE</code>	Logical value indicating whether the variable names are given on the first line of the file
<code>sep="\t"</code>	The values on each line are separated by this character ("\"t"=Tab character; ""=whitespace; ",", "="; etc.)
<code>dec=". "</code>	Decimal mark for numbers ("." or ",")
<code>row.names=1</code>	The first column of the file gives the individuals' names. If this is not the case, simply omit this argument

When using the function `read.table()`, you will need to specify the value of the argument `file` which must contain, in a character string, the name of the file and possibly its complete path. You might have noticed that we used the function `file.choose()`, which opens up a dialogue window to select a file and returns the required character string. This is an easy method to get the path to a file, but the path can also be specified explicitly:

```
my.data <- read.table(file="C:/MyFolder/data.txt")
```

Warning

Note that file paths are specified using slashes (/). This notation comes from the UNIX environment. In R, you cannot use backslashes (\), as you would in Microsoft Windows, unless you double all the backslashes (\\).



Another option is using the function `setwd()` to change the work directory (equivalent to using the menu “File/Change current directory”). The argument `file` will then accept the file name alone, without its path.

```
setwd("C:/MyFolder")
my.file <- "mydata.txt"
data <- read.table(file=my.file)
```

Your data are now available in the R console: they are stored in the object which you have chosen to call `data`. You can visualize them by typing `data`; you can also type `head(data)` or `tail(data)` to display only the beginning or the end of the data set.

Tip

- The function `attach()` (see Chap. 9) gives direct access to the variables (columns) of a `data.frame` by typing the name of a variable as it is written on the first line of the file in ASCII format (assuming this is the case).

```
attach(data)
```



- If your file contains completely empty lines, or incomplete lines, use the two arguments `fill=TRUE` and `blank.lines.skip=FALSE`.

Do it yourself



Create a folder called DataFolder. Now download the file http://www.biostatisticien.eu/springer/Intima_Media_Thickness.txt and save it in the folder DataFolder.

Use the function `readLines()` to visualize the beginning of the data file, to get an idea of how it is structured and thus to determine which arguments of the function `read.table()` you will need.

```
setwd("path/to/DataFolder/") # Replace path/to/ with your
                             path.
readLines("Intima_Media_Thickness.txt",n=5)
```

You will get the following output:

```
[1] "GENDER AGE height weight tobacco packyear SPORT measure alcohol"
[2] "1 33 170 70 1 1 0 0,52 1"
[3] "2 33 177 67 2 20 0 0,42 1"
[4] "2 53 164 63 1 30 0 0,65 0"
[5] "2 42 169 76 1 26 1 0,48 1"
```

You will notice that the first line gives the variables names. Fields are separated by simple whitespace, and the decimal mark is a comma. Therefore, you need to use the arguments `header=TRUE`, `sep=" "` and `dec=",."`.

```
mydata <- read.table("Intima_Media_Thickness.txt",sep=" ",
                      header=TRUE,dec=",")
mydata      # To display the content of mydata.
head(mydata) # Only displays the first few rows
              # of the data.frame.
```

Note that some data points are missing, as indicated by the symbol NA.

Let us now verify the structure of the object `mydata` and the types of its columns:

```
class(mydata)
str(mydata)
```

The function `attach()` is used to enable a direct access to the variables of the table.

```
attach(mydata)
```

The command `names(mydata)` outputs variable names. You can use these to make calculations with the variables, for example,

```
mean(AGE)    # Mean of age.
var(taille) # Variance of the heights.
```

Note that case (upper/lower) is important.

The function `read.table()` takes many arguments. However, since many data sets come in a standard format, a few functions exist to read these easily. Such functions are in fact equivalent to calling `read.table()` with some arguments filled in by default.

For example, if you have a file in the `.csv` format (`csv` stands for *comma-separated values*), created for example using OpenOffice's spreadsheet, you can also use the following function:

```
read.csv(file.choose()) # To read comma-separated data
# (with a . as decimal mark).
read.csv2(file.choose()) # To read semi-colon-separated data
# (with a , as decimal mark).
```

To read Tab-separated data, it is better to use

```
read.delim(file.choose()) # (with a . as decimal mark).
read.delim2(file.choose())# (with a , as decimal mark).
```

4.1.1.2 Reading Data with `read.ftable()`

Sometimes, individual data are not available: instead, we only have a contingency table. In this case, the relevant import function is `read.ftable()`.

For instance, suppose that the contents of the file `Intima_ftable.txt` come in the following form:

					alcohol	nondrinker	occasional drinker	regular drinker
		GENDER		tobacco				
"M"	"non-smoker"	6		19		7		
	"former smoker"	0		9		0		
	"smoker"	1		6		5		
"F"	"non-smoker"	12		26		2		
	"former smoker"	3		5		1		
	"smoker"	1		6		1		

The following functions can be used to read and display these data in R:

```
Intima.table <- read.ftable("Intima_ftable.txt", row.var.names
                           =c("GENDER", "tobacco"), col.vars=list("alcohol"=
                           c("nondrinker", "occasional drinker",
                           "regular drinker")))
ftable(Intima.table)
```

The output will then be

		alcohol nondrinker occasional drinker regular drinker					
		GENDER		tobacco			
"M"	"non-smoker"	6		19		7	
	"former smoker"	0		9		0	
	"smoker"	1		6		5	
"F"	"non-smoker"	12		26		2	
	"former smoker"	3		5		1	
	"smoker"	1		6		1	

See also

We shall present a descriptive analysis of this type of data in Chap. 11, pp. 345, 352 and 371. Note that standard statistical tests on contingency tables are possible, such as the chi-squared independence test presented in Chap. 13, p. 435. You may also be interested in the article <http://www.jstatsoft.org/v17/i03/paper> which presents several tools to analyse such data.

4.1.1.3 Reading Data with the Function `scan()`

The function `scan()` takes many arguments. It is useful when the data are not organized as a rectangular table. We recommend you read the documentation `help(scan)`.

For example, suppose your data file, called `Intima_Media2.txt`, contains the following lines:

File description:

The individual data are registered for nine variables
in the following order:

GENDER AGE height weight tobacco packyear SPORT measure alcohol

Data:

```
1 33 170 70 1 1 0 0,52 1 2 33 177 67 2 20 0 0,42 1
2 53 164 63 1 30 0 0,65 0 2 42 169
76 1 26 1 0,48 1
```

Here are the commands we suggest you use to read this file. The argument `skip=n` is used to omit reading the first n lines of the file.

```
# Reading variable names:
variable.names <- scan("Intima_Media2.txt",skip=4,nlines=1,what="")
# Reading data:
data <- scan("Intima_Media2.txt",skip=7,dec=",")
mytable <- as.data.frame(matrix(data,ncol=9,byrow=TRUE))
colnames(mytable) <- variable.names
```

Here is the output of variable `mytable`:

	GENDER	AGE	height	weight	tobacco	packyear	SPORT	measure	alcohol
1	1	33	170	70	1	1	0	0.52	1
2	2	33	177	67	2	20	0	0.42	1
3	2	53	164	63	1	30	0	0.65	0
4	2	42	169	76	1	26	1	0.48	1

Tip

Note that the functions `read.table()` and `scan()` can also be used to read online ASCII files directly from the Internet.

```
read.table("http://www.biostatisticien.eu/springeR/  
temperature.dat")
```



4.1.2 Importing Data from Excel or the Open Office Spreadsheet

4.1.2.1 Copy-Pasting

Using the mouse, select the range of the data (in the spreadsheet) which you wish to incorporate into R. Once the data are selected, copy them to the clipboard (from the Edit menu, or with the keyboard shortcuts **CTRL+C** on Windows or **COMMAND+C** on a Mac).

All you need to do now is type the following instructions in the R console to transfer the data from the clipboard:

```
x <- read.table(file("clipboard"), sep="\t", header=TRUE, dec=",")
```

Tip

The instruction `fix(x)` opens a small spreadsheet in R, which can be used to visualize and edit the data stored in x. It is more useful than the command `edit`, which only allows modifications. Similarly, the function `View()` displays the data in a small spreadsheet, but cannot be used to edit them.

**Warning**

Be aware that the Excel file might include formulae or other hidden characters in the data range which you wish to copy. A possible workaround is to first copy and then do a special paste of this data range in a new sheet of the Excel file. You can then use the function `read.table()` on this new sheet, as indicated above.



4.1.2.2 Using an Intermediary ASCII File

Save your file in an ASCII format, then refer to the previous section.

- Under Excel, go to **File / Save as ...** and choose **Data type: Text (tab-separated) (*.txt)**, then save.
- Under OpenOffice, go to **File / Save as ...** and choose **File type: CSV text (.csv; .txt)**, then save.
In the next window, choose:
 - field separator: Tab
 - text separator: "then click OK.

4.1.2.3 Using Specialized Packages

A few packages exist to read **.xls** files directly in R. One function worth mentioning is **read.xls()** from the package **gdata**, which works very well, as long as your computer has PERL installed (this free software can be obtained by installing the file <http://www.biostatisticien.eu/springeR/Rtools29.exe>). You can also use the package **xlsReadWrite**.

4.1.3 Importing Data from SPSS, Minitab, SAS or Matlab

The following table gives the packages and R functions you can use to import data from common proprietary software (Table 4.3).

The function **lookup.xport()** outputs (as a list) information on the SAS library of a SAS XPORT file (extension ***.xpt**).

Warning

First, note that if you use Windows, the package **foreign** is pre-installed (but not loaded) in R and that you cannot install another version from the CRAN (only Linux and Mac versions are available).



Also note the following caveats. The function **read.spss()** can require the argument **reencode="utf8"** under Linux. The function **read.mtp()** works on files containing only numeric data. At the time of writing, the function **read.xport()** cannot be used to read files directly from the Internet.

Table 4.3: Packages and R importation functions from common software

Software	Package	R function	File extension	Output format
SPSS	foreign	read.spss()	*.sav	list
Minitab	foreign	read.mtp()	*.mtp	list
SAS	foreign	read.xport()	*.xpt	data.frame
Matlab	R.matlab	readMat()	*.mat	list

4.1.4 Large Data Files

R can handle large data sets. For this, you need to specify explicitly the type of each column. If you do not, R will have to read the entire file to check that numeric columns are indeed numeric (it could be the case that a column contain numbers at the beginning, then character strings later on). The following example illustrates this point with genomic data, well known for their large size. You will need to download the 50MB file <http://www.biostatisticien.eu/springeR/dbsnp123.dat> to your computer, then try the instructions below.

Warning

Be careful that if you issue the commands below, this may freeze your R session for a few minutes.



```
tm <- Sys.time() # Gets the current time.
dbsnp <- read.table("dbsnp123.dat")
Sys.time()-tm
Time difference of 5.063645 mins

tm <- Sys.time()
dbsnp <- read.table("dbsnp123.dat", colClasses=rep("character", 3))
Sys.time()-tm
Time difference of 13.75810 secs
```

Very big data sets can thus be handled by R relatively quickly, when the correct instructions are given. The main limit is how much RAM you have available. Note also that using the function `scan()` instead of `read.table()` in the previous example would give similar execution times.

Large data sets are sometimes stored in binary format. In that case, the function `readBin()` can be used to read the data. We shall see an example in the practical of Chap. 7.

See also

If R displays a message indicating a failure of memory, you could consult with profit Sect. 9.8.



Advanced users

If the function `scan()` is used correctly, a text file can be read very quickly (as quickly as with the SAS software, for example).

 If your file is really big, you should consider storing your data in a data base (e.g., MySQL) and accessing them piece by piece. See Sect. 4.4 for more details.

Also note that a few packages exist to handle large data sets, such as `R.huge` and `filehash`. The latter is more general than the former: with the `filehash` package, the limit to the size of data which can be handled is the size of the hard disk.

— SECTION 4.2 —

Exporting Data

4.2.1 Exporting Data to an ASCII Text File

The relevant function is `write.table()`.

Suppose you have a `data.frame` called `mydata`, containing data that you wish to save in a text file. You would then use the instruction:

```
write.table(mydata, file = "myfile.txt", sep = "\t")
```

Note

 There also exists a function `write()`, which is used on vectors and matrices. This function has an interesting argument: `ncolumns` allows you to specify the number of columns in the resulting file. Note however that the file will contain the transpose of the matrix or vector you are writing.

4.2.2 Exporting Data to Excel or OpenOffice Calc

For example, type the following instructions in the R console:

```
X <- data.frame(Weight=c(80,90,75),Height=c(182,190,160))
write.table(X,file("clipboard"),sep="\t",dec=",",row.names=FALSE)
```

The data have now been copied to the clipboard. You can now paste them in your spreadsheet, for example, by typing CTRL+V.

You can also use the package `xlsReadWrite` (only under Windows).

— SECTION 4.3 —

Creating Data

4.3.1 Entering Toy Data

This section shows how you can quickly create some data. This is useful when you need to test various **R** functions on small data sets.

The main functions are `c()`, `seq()`, `:` and `rep()`:

- The function `c()` is used to create a vector by concatenating its arguments:

```
> c(1,5,8,2.3)
[1] 1.0 5.0 8.0 2.3
```

- The function `seq()` generates a sequence of values as a vector.

```
> seq(from=4,to=5)
[1] 4 5
> seq(from=4,to=5,by=0.1)
[1] 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0
> seq(from=4,to=5,length=8)
[1] 4.000000 4.142857 4.285714 4.428571 4.571429 4.714286
[7] 4.857143 5.000000
```

Note

The functions `c()` and `rep()` can also be used to create character strings.



- The function `"":()` generates a sequence of integers.

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

- The function `rep()` replicates the values of its first argument in several smart ways. We leave it to the sagacity of the reader to understand all the following instructions:

```
> rep(1,4)
[1] 1 1 1 1
> rep(1:4, 2)
[1] 1 2 3 4 1 2 3 4
```

```
> rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4
> rep(1:4, c(2,1,2,3))
[1] 1 1 2 3 3 4 4 4
> rep(1:4, each = 2, len = 4)
[1] 1 1 2 2
> rep(1:4, each = 2, len = 10)
[1] 1 1 2 2 3 3 4 4 1 1
> rep(1:4, each = 2, times = 3)
[1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

4.3.2 Generating Pseudo-Random Numbers

The function `runif()` generates a sequence of randomly generated numbers (at uniform).

```
> runif(5)
[1] 0.4344968 0.7153407 0.4561363 0.9580362 0.7260245
> runif(5,min=2,max=7)
[1] 5.634204 4.046403 5.415685 5.251441 2.209174
```

The function `rnorm()` generates a sequence of random numbers from a normal distribution.

```
> rnorm(5)
[1] 0.13585341 -0.09483162 -2.12326103 0.45974393 1.29587671
> rnorm(5,mean=2,sd=3)
[1] -0.8673785 3.5660222 0.9401026 3.4794672 4.2175481
```

See also

We shall encounter many other similar functions in Chap. 12, p. 405.

4.3.3 Entering Data from a Hard Copy

- Creating a vector with the function `scan()`

In this context, `scan()` is more user-friendly than `c()`. It can be used to easily enter data as you go.

```
> z <- scan() # R is waiting for you to enter data.
1: 4.2
2: 5.6
3: 8.9
4: 1
5: 2.3
6:      # Press ENTER after an empty line
       # to halt the procedure.
```

```
Read 5 items
> z
[1] 4.2 5.6 8.9 1.0 2.3
```

- **Creating several vectors of different lengths**

The function `data.entry()` is useful for this purpose. This function does not output anything. The variables you enter by hand are stored in the small spreadsheet which is displayed.

```
# The following instruction (which is explained later on)
# can be used to delete all the objects in the session.
rm(list=ls())
data.entry("")
```

You can now change the names of the variables (columns) and enter data. Columns can contain different numbers of observations. If you leave the mini spreadsheet and type in the instruction `ls()`, you will see the variables you have created.

Mac

The way this function works can vary on different operating systems.



- **Creating an individual×variables table**

To enter data directly into R's mini spreadsheet (as if using Excel), simply use the function `de()` (for *data entry*), as shown in the following instruction.

```
X <- as.data.frame(de(""))
```

Warning

Remember to change the names of the variables, as well as the types of the columns (`numeric` or `character`), by clicking on the cells on the first row of the table (the row with the variable names). Once you have finished entering your data, you need to close that window to return to the R console.



If you need to make small modifications to your data table X, simply use the function `fix(): fix(X)`.

Tip

The following (optional) function is used to name the rows of X:

```
rownames(X) <- paste("ind", 1:nrow(X), sep = "")
```

The names of the individuals will then appear in the first column of the mini spreadsheet (called `row.names`).



SECTION 4.4

† Reading/Writing in Databases

R is capable of communicating with most database management systems (DBMS). In this section, we briefly look at the main operations for the DBMS MySQL.

EasyPHP is an environment with two servers (an Apache web server and a MySQL database server), a script interpreter (PHP) and an SQL administration tool (phpMyAdmin). Download, install and run the latest version of EasyPHP (<http://www.easypg.org>).

Note

You may need to configure your firewall so that it allows services started by EasyPHP (mysqld and Apache).

4.4.1 Creating a Database and a Table

Start EasyPHP and right-click on the icon  on the right side of the task bar (you may need to check on the small white triangle **Show hidden icons**), then select **Administration**. Your web browser should then open (if it does not, try using the browser **firefox** and configuring Apache by right-clicking on the EasyPHP icon and then **Configuration: Listen 127.0.0.1:80**). On the page that opens up, check on the **open** button in section **MODULES** to get to the phpMyAdmin administration page. Then click on the tab **Databases** and create a new database called **BMI**. Click on the icon **Create table** which appears in the left panel.

Then give a name to the table: **mytable**.

Define four fields for your table (one per line), and fill them as follows:

- **Name** = FirstName, **Type** = VARCHAR and **Length/Values** = 20;
- **Name** = Weight, **Type** = FLOAT and **Length/Values** = 3;
- **Name** = Height, **Type** = FLOAT and **Length/Values** = 3;
- **Name** = BMI, **Type** = FLOAT and **Length/Values** = 5.

Click on **Save**.

4.4.2 Creating a Data Source Compatible with MySQL

The functions `odbcConnect()`, `sqlQuery()` and `odbcClose()` from package `RODBC` are useful to handle databases from different systems (PostgreSQL, MySQL,

etc.) through R, thanks to an ODBC (*open database connectivity*) link created with an existing database. Under Windows, here is how you create an ODBC data source compatible with MySQL.

First install MyODBC (MySQL Connector/ODBC) (<http://dev.mysql.com/downloads/connector/odbc>). Next, in order to display the window data source (ODBC), you will have to execute the file C:\Windows\System32\odbcad32.exe.

Tip

For a 64-bit computer, you may need to use file C:\Windows\Syswow64\odbcad32.exe.



Click on **Add** and select entry **MySQL ODBC 5.1 Driver**. In the windows which opens up, enter the following fields:

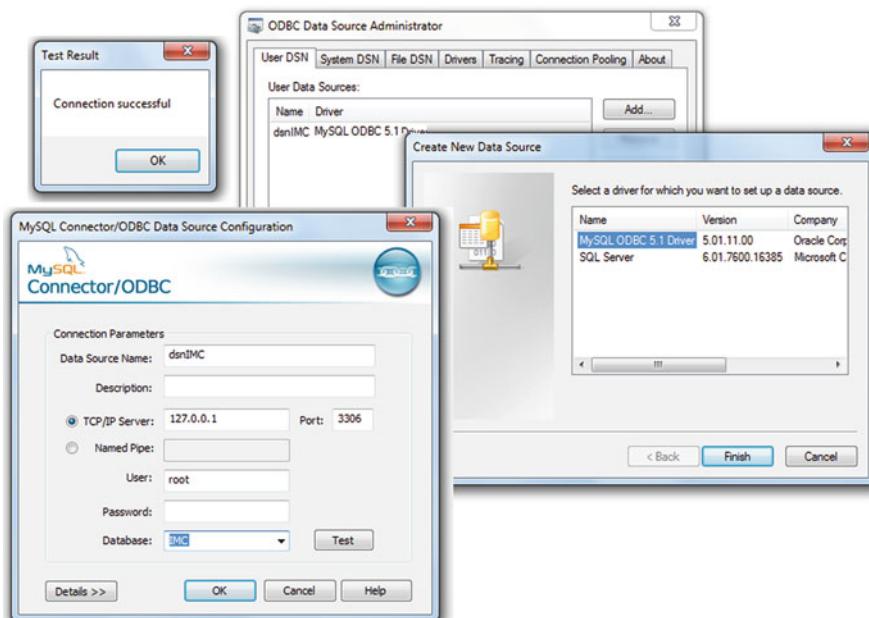
Data Source Name: dsnBMI

TCP/IP Server: 127.0.0.1 **Port:** 3306

User: root

Database: BMI

Then click on button **Test**. The message **Connection successful** should appear if everything happened correctly. Then click on **OK** (twice) to close the dialogue boxes.



We have thus configured an ODBC link which will allow R to communicate with MySQL.

Linux

Check that the file `/etc/odbcinst.ini` includes references to the drivers of the MySQL database. You also need to modify (as root) the file `/etc/odbc.ini` so that it includes the following lines:



```
[dsnBMI]      # name of the data source.
Description =
Driver = MySQL
Server = localhost
Database = BMI
Port = 3306
```

4.4.3 Writing in a Table

We shall now write information in table `mytable` of base `BMI`. The following instructions are used to add the weight, height and BMI (temporarily set at 0) for an individual named Peter:

```
> require("RODBC")
> Connection <- odbcConnect(dsn="dsnBMI",uid="root",pwd="")
> request <- "INSERT INTO mytable VALUES ('Peter',72,182,0)"
> result <- sqlQuery(Connection,request)
> odbcClose(Connection)
```

Here is now a multiple insertion example:

```
> FirstNames <- c("Ben","John")
> Weight <- c(70,75)
> Height <- c(190,184)
> BMI <- round(Weight/(Height/100)^2,3)
> mat <- cbind(FirstNames,Weight,Height,BMI)
> insertmult <- function(vect)
+   paste(",toString(c(encodeString(vect[1],quote='')),",
+   vect[-1])),")",sep="")
> tobeinserted <- toString(apply(mat,1,insertmult))
> tobeinserted
[1] "('Ben', 70, 190, 19.391),('John', 75, 184, 22.153)"
> require("RODBC")
> Connection <- odbcConnect(dsn="dsnBMI",uid="root",pwd="")
> request <- paste("INSERT INTO mytable (FirstName,Weight,
+                   Height,BMI)VALUES ",tobeinserted,sep="")
> result <- sqlQuery(Connection,request)
> odbcClose(Connection)
```

Tip

You can return to phpMyAdmin to check that the table `mytable` has indeed been modified (tab **Browse**).



4.4.4 Reading a Table

To read from R information in the table `mytable`, you can use the following instructions:

```
> require("RODBC")
> Connection <- odbcConnect(dsn="dsnBMI",uid="root",pwd="")
> request <- "SELECT * FROM mytable"
> data <- sqlQuery(Connection,request)
> odbcClose(Connection)
> data
   FirstName    Weight    Height     BMI
1 Peter        72       182      0.000
2 Ben          70       190      19.391
3 John         75       184      22.153
```

Memorandum

`read.table()`: read a rectangular data file
`scan()`: read data line by line
`read.ftable()`: read a contingency table
`ftable()`: display a contingency table
`readLines()`: read and display a few lines of a file
`file.choose()`: open a dialogue window to select a file
`file, header, sep, dec, row.names, skip`: main arguments of `read.table()`
`read.spss()`, `read.mtp()`, `read.xport()`, `readMat()`: import data from other software
`write.table()`: write a data file
`file("clipboard")`: copy from or paste to the clipboard
`c()`: create a sequence of elements of the same nature
`seq()`: create a sequence of numbers or a character string
`rep()`: repeat the values of the first argument
`de()`, `data.entry()`: enter data using a mini spreadsheet
`fix()`: modify a data.frame or a matrix in a mini spreadsheet



Exercises

- 4.1-** Name the three main R functions to import data from an ASCII text file.
- 4.2-** One of the usual data reading functions takes the following arguments: `header`, `sep`, `dec`, `row.names`, `skip`, `nrows`. Explain their purpose. Give an example of a value each argument can take.
- 4.3-** What is the purpose of the function `readLines()`?
- 4.4-** What is the purpose of the function `fix()`?
- 4.5-** Give the specificities of the functions `read.csv()`, `read.csv2()`, `read.delim()` and `read.delim2()`.
- 4.6-** What is the purpose of the function `read.ftable()` ?
- 4.7-** What is the difference between the functions `scan()` and `read.table()` ?
- 4.8-** Explain how you would import data from an Excel spreadsheet. Give details.
- 4.9-** Which package includes several functions to import data from commercial statistical software?
- 4.10-** When reading a large data file, which argument to the function `read.table()` can speed up the reading?
- 4.11-** Which R function should be used to write to a file a data set contained in a `data.frame`? Which other function do you know?
- 4.12-** Name the four basic functions to create a vector.
- 4.13-** Explain how the function `seq()` can be used to get the following vector:
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
- 4.14-** Give the shortest R instruction which outputs the following vector:

1 1 2 2 3 3

- 4.15-** Give the shortest R instruction which outputs the following vector:

1 2 3 1 2 3

- 4.16-** Name two R functions which can be used to enter data by hand in a mini spreadsheet.

Worksheet

Reading Various Data Sets

A- Entering Data from a Hard Copy

- *Cold sore*: A total of 30 patients have been randomly assigned to one of five treatments against cold sore, including one placebo (there are six patients in each treatment group). For each patient, the number of days between the apparition of the first blisters and complete healing has been recorded.

Treatments					
	trt1 (placebo)	trt2	trt3	trt4	trt5
5		4	6	7	9
8		6	4	4	3
7		6	4	6	5
7		3	5	6	7
10		5	4	3	7
8		6	3	5	6

We would like to know whether there is a difference between the treatments by comparing the mean healing time in each independent random sample (treatment group). The relevant statistical method is called ANOVA; we shall present it in Chap. 15. In this practical, we shall simply see how to enter these data in R to compute the sample mean for each treatment.

- 4.1-** Enter the data in R directly, using the function `de()`.
- 4.2-** Use the function `attach()` and then the function `mean()` to compute the mean for each treatment.
- 4.3-** Compute the means of all treatments simultaneously, thanks to the function `colMeans()`.
- 4.4-** Use the function `write.table()` to save your `data.frame` in a file called `blisters.txt`.
- 4.5-** Open your file in a text editor and check that there was no problem.
- 4.6-** Use the function `rm()` to delete all the R objects you have created in your work environment.
- 4.7-** Import the file `blisters.txt` with the function `read.table()` and display the data.

- *Risk factors for atherosclerosis:* As part of a study on risk factors for atherosclerosis, data were collected and are summed up in this contingency table:

		alcohol	nondrinker	occasional-drinker	regular-drinker
GENDER		tobacco			
M	non-smoker	6		19	7
	former smoker	0		9	0
	smoker	1		6	5
F	non-smoker	12		26	2
	former smoker	3		5	1
	smoker	1		6	1

It would be interesting to know whether there is a dependence between smoking and drinking, according to gender. To enter these data in R, there are several steps:

4.1- Use the function `scan()` to get a matrix X of size 6×3 . This matrix will contain the data only.

4.2- Use the instruction `class(X) <- "ftable"` to specify that it is a contingency table.

4.3- Type the two instructions:

```
attributes(X)$col.vars <- list(alcohol=c("nondrinker",
                                         "occasional-drinker", "regular-drinker"))
attributes(X)$row.vars <- list(GENDER=c("M", "F"), tobacco=
                                c("non-smoker", "former smoker", "smoker"))
```

4.4- Display the contingency table you have created.

4.5- Use the function `write.ftable()` to save the contingency table in a file called `athero.txt`.

4.6- Open the file in a text editor and check that there was no problem.

4.7- Use the function `rm()` to delete all the R objects you have created in your work environment.

4.8- Import the file `athero.txt` with the function `read.table()` and display the data.

B- Importing from Other Software

During a study of BMI (body mass index) of children, a team of statisticians collected data in different formats. As an exercise, we are going to read these various formats. There are several files called `bmichild`, but with different file extensions:

4.1- Import the file `bmichild.xls` into a data.frame called `bmi.XLS`.

4.2- Import the file `bmichild.xpt` into a data.frame called `bmi.SAS`.

4.3- Import the file `bmichild.sav` into a data.frame called `bmi.SPSS`.

4.4- Import the file `bmichild.mat` into a data.frame called `bmi.MAT`. The procedure is trickier for this file, so here are detailed instructions:

```
x <- readMat("bmichild.mat")
class(x) # x is a list
x       # you can see that the data are in $bmi[,1]
x <- x$bmi[,1]
# Note that the elements of GENDER and zep
```

```
# are recorded in a list.  
x$GENDER  
class(x$GENDER) <- "character"  
x$GENDER  
class(x$zep) <- "character"  
bmi.MAT <- as.data.frame(x)
```

- 4.5-** To check that there was no problem during importation, use the function `summary()` on all these data.frames. This will display a few numerical summaries.
- 4.6-** All these data.frames are identical. Save one of them in a file called `bmichild.txt`.

C- Importing More Complex Data Files

Statisticians often encounter data files in non-standard formats. This section therefore provides training in reading several non-standard files on which we wish to perform statistical analysis.

- 4.1-** Import the file `raf98.gra` into the most relevant structure. To this end, you will need to read the associated file `geoidformat.txt` which describes the file format.
- 4.2-** Import the file `Infarction.xls` into a data.frame. Make sure you handle missing values correctly.
- 4.3-** The file `nutrition_elderly.txt` contains 13 variables measured on 226 individuals. Import the file into a data.frame (hint: use the functions `t()` and `as.data.frame()`).
- 4.4-** The file `Birth_weight.txt` contains ten variables measured on 189 individuals. Import it into a data.frame, which will contain the names of the variables as well as the names of the individuals (these are available in the column `Id`). Remember that you can use the online help!

Chapter 5

Data Manipulation, Functions

Prerequisites and goals

- First, read Chaps. 3 and 4.
- In this chapter, we shall present elementary data manipulation functions. We shall also describe the main control structures and show how to use the extraction tool of the components of an object. This is a very powerful method, which you will need in order to use R in the most efficient way. We shall present direct extraction and extraction by logical mask. We shall also explain how to handle character strings and dates in R.

SECTION 5.1

Operations on Vectors, Matrices and Lists

5.1.1 Vector Arithmetic

One of the advantages of R is that it can operate on vectors and matrices. For example, the third instruction below

```
> x <- c(1,2,4,6,3)
> y <- c(4,7,8,1,1)
> x+y
[1] 5 9 12 7 4
```

returns, in a single operation, the vector of sums $(x_1 + y_1, \dots, x_n + y_n)$.

Warning

This is one of the **main strengths** of R. It is called vectorization. You should get used to working in this fashion. Thus, you should **avoid using programming loops**, as is often done in other languages: such code would run much slower.

R operates in a similar fashion for many functions, such as `+`, `*`, `-`, `/`, `exp`, `log`, `sin`, `cos`, `tan`, `sqrt` and so on.

For example, the following instruction calculates the exponential of all the elements of the matrix M:

```
> M <- matrix(1:9,nrow=3)
> exp(M)
[,1]      [,2]      [,3]
[1,] 2.718282 54.59815 1096.633
[2,] 7.389056 148.41316 2980.958
[3,] 20.085537 403.42879 8103.084
```

5.1.2 Recycling

At this stage, it is important to note how R behaves when given an operation on two vectors of different lengths. R will complete the shortest vector, reusing the values of this vector. The following example should help understand this concept:

```
> x <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15) # Vector of length
# 15.
> y <- c(1,2,3,4,5,6,7,8,9,10)                 # Vector of length
# 10.
> x+y                                         # Vector of length
# 15.
[1]  2  4  6  8 10 12 14 16 18 20 12 14 16 18 20
```

R has completed the vector y thus `c(1,2,3,4,5,6,7,8,9,10,1,2,3,4,5)` by reusing its values, in a circular fashion.

Note

This behaviour is called **recycling**. It is important that you be aware of this behaviour, since it can provoke hard-to-detect errors. As a matter of fact, R usually displays a warning:

Warning message:

In x + y :

*the length of the longest object is not a multiple
of the length of the shortest object*



Here is another example of recycling, this time used to create a matrix. The vector 1:4 is reused in a circular fashion to fill in the matrix, which is declared to be of size 3×3 .

```
> matrix(1:4, ncol=3, nrow=3)
[,1] [,2] [,3]
[1,]    1    4    3
[2,]    2    1    4
[3,]    3    2    1
```

5.1.3 Basic Functions

Here are a few basic data manipulation functions. These are used very often; it is essential that you know them.

- `length()`: returns the length of a vector.

```
> length(c(1,3,6,2,7,4,8,1,0))
[1] 9
```

- `sort()`: sorts the elements of a vector, in increasing or decreasing order.

```
> sort(c(1,3,6,2,7,4,8,1,0))
[1] 0 1 1 2 3 4 6 7 8
> sort(c(1,3,6,2,7,4,8,1,0), decreasing=TRUE)
[1] 8 7 6 4 3 2 1 1 0
```

- `rev()`: rearranges the elements of a vector in reverse order.

```
> rev(c(1,3,6,2,7,4,8,1,0))
[1] 0 1 8 4 7 2 6 3 1
```

- `order()`, `rank()` : the first function returns the vector of (increasing or decreasing) ranking indices of the elements. The second function returns the vector of ranks of the elements. In case of a tie, the ordering is always from left to right.

```
> vec <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
> names(vec) <- 1:9
```

```
> vec
1 2 3 4 5 6 7 8 9
1 3 6 2 7 4 8 1 0
> sort(vec)
9 1 8 4 2 6 3 5 7
0 1 1 2 3 4 6 7 8
> order(vec)
[1] 9 1 8 4 2 6 3 5 7
> rank(vec)
 1   2   3   4   5   6   7   8   9
2.5 5.0 7.0 4.0 8.0 6.0 9.0 2.5 1.0
```

- `unique()`: as the name suggests, this function removes the duplicates of a vector.

```
> unique(c(1,3,6,2,7,4,8,1,0))
[1] 1 3 6 2 7 4 8 0
```

- `duplicated()`: indicates elements which have already been encountered earlier in the vector (read from left to right).

```
> duplicated(c(1,3,6,2,7,4,8,1,0))
[1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE
```

5.1.4 Operations on Matrices and Data.Frames

We shall describe several specialized R functions which give information on a matrix (or a data.frame) or help manipulate its rows and columns.

See also

 Standard matrix operations (product, decomposition, Jacobian, ...) are described in Chap. 10, page 316.

5.1.4.1 Information on Architecture

Here are a few functions which give information on a matrix or a data.frame:

- `dim()`: size of the matrix or data.frame
- `nrow()`: number of rows
- `ncol()`: number of columns
- `dimnames()`: names of rows and columns (as a list)
- `names(), colnames()`: names of columns
- `rownames()`: names of rows

Do it yourself

Import, into an R object called X, the data from the file http://www.biostatisticien.eu/springeR/Weight_birth.xls, and use the above functions on X. Note that the first column of the file gives the patient identifier.

5.1.4.2 Merging Tables

It is often very useful to combine (merge) several matrices or data.frames. The basic functions to this end are cbind() to merge columns and rbind() to merge rows.

- **Merging columns**

The generic function is cbind().

```
> cbind(1:4,5:8)
 [,1] [,2]
 [1,]    1    5
 [2,]    2    6
 [3,]    3    7
 [4,]    4    8
```

However, this function is not optimal, as the following example shows. Let us try to merge in columns the two following tables:

	Id	GENDER	Weight		Id	GENDER	Height
X1=	1	M	75		1	M	182
	2	F	68	U	2	F	165
	3	F	48	X2=	3	F	160
	4	M	72		4	M	178

```
> X1 <- data.frame(Id=1:4,GENDER=c("M","F","F","M"),
+                     Weight=c(75,68,48,72))
> X2 <- data.frame(Id=1:4,GENDER=c("M","F","F","M"),
+                     Height=c(182,165,160,178))
> cbind(X1,X2)
   Id GENDER Weight Id GENDER Height
1   1       M     75  1       M    182
2   2       F     68  2       F    165
3   3       F     48  3       F    160
4   4       M     72  4       M    178
```

This works, but it is a shame that the columns `Id` and `GENDER` are duplicated. A very useful function in this context is `merge()`:

```
> merge(X1,X2)
  Id GENDER Weight Height
1  1      M     75    182
2  2      F     68    165
3  3      F     48    160
4  4      M     72    178
```

Now suppose that the individuals are not sorted in the same way in both tables.

	Id	GENDER	Weight		Id	GENDER	Height
X1=	1	M	75	U	2	F	165
	2	F	68		3	M	182
	3	F	48		4	M	178
	4	M	72		3	F	160

In this case, you cannot use the function `cbind()`, but the function `merge()` still works:

```
> X3 <- data.frame(Id=c(2,1,4,3),GENDER=c("F","M","M","F"),
+                     Height=c(165,182,178,160))
> merge(X1,X3)
  Id GENDER Weight Height
1  1      M     75    182
2  2      F     68    165
3  3      F     48    160
4  4      M     72    178
```

You will have noticed that, by default, the function `merge()` combines two data.frames. Let `X` and `Y` be the two data.frames we wish to merge, and let `Z` be the data.frame resulting from the merge of `X` and `Y`. The merge is based upon the columns of these two data.frames which have the same names. These columns will be called “common columns”. The argument `by` can be used to specify (force) which columns are common. The value of this argument can be a vector of names, a vector of indices or a vector of logical values. All other columns will then be treated as different columns by `merge()`, even if they bear the same name. The function `merge()` then works in the following way:

- For every row (individual) of the data.frame `X`, the function `merge()` compares the elements of this row to those of every row of `Y`, but only over the subset of common columns

- If it finds a perfect match, it considers that it is the same individual: this individual is added to Z, then completed with the values from the non-common columns of X, then with the values from the non-common columns of Y.
- If no perfect match is found, the individual is either added to Z and completed with NA's (if the argument `all()` takes the value TRUE) or removed (if the argument `all()` takes the value FALSE, which is the default value).
- The operation is repeated for the next row, until the last row.

This example should help clarify things:

```
> X <- data.frame(GENDER=c("F","M","M","F"),Height=c(165,182,
+           178,160),Weight=c(50,65,67,55),Income=c(80,90,60,50))
> Y <- data.frame(GENDER=c("F","M","M","F"),Height=c(165,182,
+           178,160),Weight=c(55,65,67,85),Salary=c(70,90,40,40),
+           row.names=4:7)
> X
  GENDER Height Weight Income
1      F     165      50     80
2      M     182      65     90
3      M     178      67     60
4      F     160      55     50
> Y
  GENDER Height Weight Salary
4      F     165      55     70
5      M     182      65     90
6      M     178      67     40
7      F     160      85     40
> merge(X,Y,by=c("GENDER","Weight"))
   GENDER Weight Height.x Income Height.y Salary
1      F      55     160     50     165     70
2      M      65     182     90     182     90
3      M      67     178     60     178     40
> merge(X,Y,by=c("GENDER","Weight"),all=TRUE)
   GENDER Weight Height.x Income Height.y Salary
1      F      50     165     80       NA      NA
2      F      55     160     50     165     70
3      F      85       NA      NA     160     40
4      M      65     182     90     182     90
5      M      67     178     60     178     40
```

Warning

You will have noticed that, by default, the function `merge()` does not take into account the names of the individuals in the data.frames X and Y, when determining the common individuals. The names of the individuals can be included either by adding a column Id to X and Y to identify the individuals or by using the name "row.names" as the value of the argument by:

```
> merge(X,Y,by=c("row.names","Weight"))
   Row.names Weight GENDER.x Height.x Income GENDER.y Height.y
1             4      55       F     160      50       F     165
   Salary
1      70
> merge(X,Y,by=c("row.names","Weight"),all=TRUE)
   Row.names Weight GENDER.x Height.x Income GENDER.y Height.y
1             1      50       F     165      80    <NA>     NA
2             2      65       M     182      90    <NA>     NA
3             3      67       M     178      60    <NA>     NA
4             4      55       F     160      50       F     165
5             5      65    <NA>     NA     NA       M     182
6             6      67    <NA>     NA     NA       M     178
7             7      85    <NA>     NA     NA       F     160
   Salary
1      NA
2      NA
3      NA
4      70
5      90
6      40
7      40
```



- **Merging lines**

The generic function is `rbind()`.

```
> rbind(1:4,5:8)
   [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

The function `smartbind()` from the package `gtools` is more sophisticated, as shown in the following example:

```
> require("gtools")
> df1 <- data.frame(A=1:5, B=LETTERS[1:5])      # The square
                                                # brackets [] to
                                                # extract
                                                # elements are
                                                # described in
                                                # section 5.5.
> df2 <- data.frame(A=6:10, E=letters[1:5])
> smartbind(df1, df2)
   A     B     E
1.1  1     A <NA>
1.2  2     B <NA>
```

```

1.3 3      C <NA>
1.4 4      D <NA>
1.5 5      E <NA>
2.1 6 <NA>   a
2.2 7 <NA>   b
2.3 8 <NA>   c
2.4 9 <NA>   d
2.5 10 <NA>  e

```

Tip

The package `gdata` includes several very interesting functions to manipulate data.



5.1.4.3 The Function `apply()`

An oft-used function is `apply()`, which applies a given function (specified as the value of the argument `FUN`) to all rows (`MARGIN=1`) or to all columns (`MARGIN=2`) of a matrix or `data.frame`.

```

> X <- matrix(c(1:4, 1, 6:8), nr = 2)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    7
[2,]    2    4    6    8
> apply(X, MARGIN=1, FUN=mean)
[1] 3 5
> apply(X, MARGIN=2, FUN=sum)
[1] 3 7 7 15

```

Tip

When the operation is summing or calculating the means of rows or columns, other possible functions are `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()`.



Do it yourself



We are going to see how to calculate the sum of squares of all rows of a matrix. First, create a matrix `M` of size 5×2 containing numbers of your choosing. Next, use the function `apply()` on the rows of the matrix `M`. You will take the argument and associated value `FUN=function(x) {sum(x^2)}`.

Warning

 In this do it yourself, we saw in passing how to use a self-created function (`sum(x^2)`) when calling `apply()`, by using the reserved word `function`. In Chap. 8, we shall explore in further detail how to create more elaborate functions.

5.1.4.4 The Function `sweep()`

The function `sweep()` is very useful. It is used to “sweep out” (in a sense defined by the value of the argument `FUN`) a certain statistic (given by the value of the argument `STATS`) from every row (`MARGIN=1`) or from every column (`MARGIN=2`) of a table. The two next examples should help you understand this function.

```
> X
      [,1] [,2] [,3] [,4]
[1,]     1     3     1     7
[2,]     2     4     6     8
> # Subtract 3 from row 1, and 5 from row 2.
> sweep(X,MARGIN=1,STATS=c(3,5),FUN="-")
      [,1] [,2] [,3] [,4]
[1,]    -2     0    -2     4
[2,]    -3    -1     1     3
> # Divide the first two columns by 2, and the last two columns
# by 3.
> sweep(X,MARGIN=2,STATS=c(2,2,3,3),FUN="/")
      [,1] [,2]      [,3]      [,4]
[1,]  0.5  1.5  0.3333333  2.333333
[2,]  1.0  2.0  2.0000000  2.666667
```

5.1.4.5 The Function `stack()`

The function `stack()` concatenates into a single vector the values of certain columns of a data.frame. This function outputs a data.frame, with the stacked vector in its first column and a second column containing a factor which indicates the origin of each observation. The function `unstack()` performs the reverse operation. This function is very useful for analysis of variance (ANOVA).

```
> X <- data.frame(trt1=c(1,6,3,5),trt2=c(8,8,3,1))
> X
  trt1 trt2
1     1     8
2     6     8
3     3     3
4     5     1
> stack(X)
  values   ind
```

```

1      1 trt1
2      6 trt1
3      3 trt1
4      5 trt1
5      8 trt2
6      8 trt2
7      3 trt2
8      1 trt2

```

5.1.4.6 The Function `aggregate()`

The function `aggregate()` splits a `data.frame` into subpopulations according to a factor (specified by the argument `by`) and applies a given function to each subpopulation.

```

> X<-data.frame(Weight=c(80,75,60,52),Height=c(180,170,165,150),
+                 Cholesterol=c(44,12,23,34),
+                 Gender=c("Male","Male","Female","Female"))
> X
  Weight Height Cholesterol Gender
1     80     180           44   Male
2     75     170           12   Male
3     60     165           23 Female
4     52     150           34 Female
> aggregate(X[,-4],by=list(Gender=X[,4]),FUN=mean)
  Gender Weight Height Cholesterol
1 Female    56.0   157.5        28.5
2   Male    77.5   175.0        28.0

```

Note

The instruction `X[,-4]` is used to extract all the columns of `X` except the fourth column. Extraction instructions are explored in further detail in Sect. 5.4.



5.1.4.7 The Function `transform()`

This function makes transformations on the columns of a `data.frame`. For example, the following example transforms the height in centimetres into the height in metres and adds to the `data.frame` a column with the BMI.

```

> X <- transform(X,Height=Height/100,BMI=Weight/(Height/100)^2)
> X
  Weight Height Cholesterol Gender      BMI
1     80   1.80           44   Male 24.69136
2     75   1.70           12   Male 25.95156
3     60   1.65           23 Female 22.03857
4     52   1.50           34 Female 23.11111

```

See also

The package `plyr` can be used to manipulate data tables in a simple and efficient way.

5.1.5 Operations on Lists

The functions `lapply()` and `sapply()` are similar to the function `apply()`: they apply a function to every element of a list. The former outputs a list; the latter outputs a vector if possible.

```
> x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,
+                                FALSE,TRUE))
> lapply(x,mean) # Mean of each element of the list.
$a
[1] 5.5
$beta
[1] 4.535125
$logic
[1] 0.5
> lapply(x,quantile,probs=(1:3)/4) # Median and quartiles of the
# elements of the list.
$a
 25% 50% 75%
3.25 5.50 7.75
$beta
 25%      50%      75%
0.2516074 1.0000000 5.0536690
$logic
25% 50% 75%
0.0 0.5 1.0
> sapply(x, quantile) # Quantiles of the elements of the list.
      a        beta  logic
0%    1.00  0.04978707  0.0
25%   3.25  0.25160736  0.0
50%   5.50  1.00000000  0.5
75%   7.75  5.05366896  1.0
100% 10.00 20.08553692  1.0
> i36 <- sapply(3:6, seq) # Creates a list of vectors.
> i36
[[1]]
[1] 1 2 3
[[2]]
[1] 1 2 3 4
[[3]]
[1] 1 2 3 4 5
[[4]]
[1] 1 2 3 4 5 6
```

```
> sapply(i36, sum) # Sum of every vector in the list.  
[1] 6 10 15 21
```

Tip

The function `do.call()` takes two arguments: the first is the name of a function and the second is the name of a list. It executes the function, taking as input the elements of the list. We shall see an example of use in the practical. We should also mention the existence of the function `mapply()`.

**SECTION 5.2**

Logical and Relational Operations

The two logical values are `TRUE` (or `T`) and `FALSE` (or `F`). Also note that in R, `NA` is considered to be a logical constant. Logical vectors are very useful in R, for example, for **extracting elements by logical mask**, as we shall see later on.

Table 5.1 on next page gives operators and functions which take logical values as input or output.

Warning

Note that the two following instructions give different results:

```
> all.equal(0.2-0.1,0.3-0.2)  
[1] TRUE  
> (0.2-0.1) == (0.3-0.2)  
[1] FALSE
```



This stems from the fact that a computer has a limited precision for its calculation. The function `all.equal()` takes an optional tolerance argument for rounding errors. We shall discuss this further in Sect. 5.7.1. See also Sect. 5.9 that explains why the second instruction above returns `FALSE`.

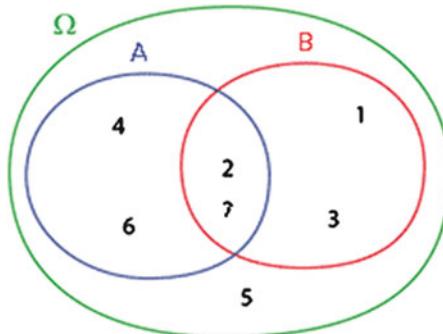
Table 5.1: Operators and functions which take logical values as input or output

Operator in R	Description	Example	Output
<code>logical()</code>	Create a vector of logical values	<code>logical(2)</code>	F F
<code>as.logical()</code>	Transform into logical values	<code>as.logical(c(0,1))</code>	F T
<code>is.logical()</code>	Is the argument a logical value?	<code>is.logical(F)</code>	T
<code>x < y</code>	Is $x_i < y_i$?	<code>c(1,4)<c(2,3)</code>	T F
<code>x > y</code>	Is $x_i > y_i$?	<code>c(1,4)>c(2,3)</code>	F T
<code>x <= y</code>	Is $x_i \leq y_i$?	<code>c(1,4)<=c(1,3)</code>	T F
<code>x >= y</code>	Is $x_i \geq y_i$?	<code>c(1,4)>=c(1,3)</code>	T T
<code>x == y</code>	Is $x_i = y_i$?	<code>c(1,4)==c(1,3)</code>	T F
<code>x != y</code>	Is $x_i \neq y_i$?	<code>c(1,4)!=c(1,3)</code>	F T
<code>!x</code>	Negation of x	<code>!c(T,F)</code>	F T
<code>x & y</code>	Term-by-term conjunction (AND)	<code>c(T,T) & c(T,F)</code>	T F
<code>x && y</code>	Sequential conjunctions	<code>F && T && T</code>	F
<code>x y</code>	Term-by-term disjunction (OR)	<code>c(T,T) c(T,F)</code>	T T
<code>x y</code>	Sequential disjunction (OR)	<code>F T F</code>	T
<code>xor()</code>	Exclusive disjunction (XOR)	<code>xor(c(T,T),c(T,F))</code>	F T
<code>any()</code>	TRUE if at least one of the x_i is TRUE	<code>any(c(T,F))</code>	T
<code>all()</code>	TRUE if all of the x_i are TRUE	<code>all(c(T,F))</code>	F
<code>all.equal()</code>	Is $x_i \approx y_i$? (see the argument tolerance)	<code>all.equal(0.2-0.1,0.3-0.2)</code>	T
<code>identical()</code>	TRUE if $\forall i, x_i = y_i$	<code>identical(1,as.integer(1))</code>	F

SECTION 5.3

Operations on Sets

R can handle all usual operations on sets (Table 5.2).



```
> A <- c(4,6,2,7) # A first set.
> B <- c(2,1,7,3) # A second set.
> vec <- c(2,3,7) # A few elements.
```

Table 5.2: Operations on sets

Operation	R Instruction	Output
Membership: $a \in A$	is.element(vec, A)	T F T
Inclusion (subset): $A \subset B$	all(A %in% B)	F
Superset: $A \supset B$	all(B %in% A)	F
Intersection: $A \cap B$	intersect(A,B)	2 7
Union: $A \cup B$	union(A,B)	4 6 2 7 1 3
Complement: $A \setminus B$	setdiff(A,B)	4 6
Symmetric difference: $(A \cup B) \setminus (A \cap B)$	setdiff(union(A,B), intersect(A,B))	4 6 1 3

Tip

It is very easy in R to define its own set functions, such as functions of inclusion, containment and symmetric difference.

```
> "%subset%" <- function(A,B) all(A %in% B)
> "%superset%" <- function(A,B) B %subset% A
> "%symdiff%"<-function(A,B) setdiff(union(A,B), intersect(A,B))
```

**SECTION 5.4****Extracting and Inserting Elements**

In this section, we shall see how to extract part of a vector, matrix or list. Indeed, R includes very specific mechanisms to this effect, which can be confusing at first, but are very powerful tools.

5.4.1 Extracting from/Inserting into Vectors

- Extraction

To extract components from a vector, use the function `"["]()`. It can take the following arguments:

- A vector of indices of elements to extract
- A vector of indices of elements not to extract
- A vector of logical values TRUE/FALSE indicating which elements to extract

A few examples will make this easier to understand.

```
> vec <- c(2,4,6,8,3)
> vec[2]
[1] 4
> "["(vec,2)           # Note: "[" is indeed a function.
[1] 4
> vec[-2]              # All elements except the second.
[1] 2 6 8 3
> vec[2:5]
[1] 4 6 8 3
> vec[-c(1,5)]
[1] 4 6 8
> vec[c(T,F,F,T,T)] # Extraction by logical mask.
[1] 2 8 3
> vec>4
[1] FALSE FALSE  TRUE  TRUE FALSE
> vec[vec>4]          # Extraction by logical mask.
[1] 6 8
```

Warning

It is important to note here the syntactical simplicity of an instruction such as `x[y>0]`, which extracts from `x` all elements of index i such that $y_i > 0$.



```
> x <- 1:5
> y <- c(-1,2,-3,4,-2)
> x[y>0]
[1] 2 4
```

You need to learn to use as often as possible such constructions, which are called **logical masks**. There are two advantages: the code is easy to read and execution is very fast.

Also note the functions `which()`, `which.min()` and `which.max()`, which are often very useful.

```
> mask <- c(TRUE, FALSE, TRUE, NA, FALSE, FALSE, TRUE)
> which(mask) # Outputs the indices corresponding to the values
# TRUE.
[1] 1 3 7
> x <- c(0:4, 0:5, 11)
> which.min(x) # Outputs the index of the smallest value.
[1] 1
> which.max(x) # Outputs the index of the largest value.
[1] 12
```

Warning

It is important to note that R does not handle index 0, unlike some other programming languages.



- **Replacement**

Replacing elements in a vector is done in a similar fashion to extraction. All you need to do is select the elements as if you wanted to extract them, then use the affectation symbol `<-` followed by the replacement elements. Of course, you need to specify the same number of replacement elements as of selected elements.

Let us examine a few examples of this principle.

```
> z
[1] 0 0 0 2 0
> z[c(1,5)] <- 1
> z
[1] 1 0 0 2 1
> z[which.max(z)] <- 0
> z
[1] 1 0 0 0 1
> z[z==0] <- 8 # The  $z_i$  such that
#  $z_i$  is worth 0 are replaced with
# 8.
> z
[1] 1 8 8 8 1
```

- **Insertion**

Inserting or adding elements to a pre-existing vector uses the function `c()`.

```
> vecA <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
> vecA
[1] 1 3 6 2 7 4 8 1 0
> (vecB <- c(vecA, 4, 1))
[1] 1 3 6 2 7 4 8 1 0 4 1
> (vecC <- c(vecA[1:4], 8, 5, vecA[5:9]))
[1] 1 3 6 2 8 5 7 4 8 1 0
```

This mechanism provides the ability to complete a vector whose size is not fixed at first.

```
> a <- c()
> a <- c(a,2)
> a <- c(a,7)
> a
[1] 2 7
```

Do it yourself



- Create the vector `height <- c(182,150,160,140.5,191)` and the vector `gender <- c(0,1,1,1,0)` containing the height (in cm) and the gender (coded as 0=M/1=F) of five people. Extract from the vector `height` the heights of the men. Use the method of extracting by indices, then repeat the task with a logical mask.

- Extract from the following vector all numbers between 2 and 3:

```
> x <- c(0.1,0.5,2.1,3.5,2.8,2.7,1.9,2.2,5.6)
```

5.4.2 Extracting from/Inserting into Matrices

• Extraction

Two methods are possible to extract elements from a matrix `X`. Each method has its own syntax.

- (a) *Extracting by indices*: `X[indr,indc]`, where `indr` is the vector of indices of rows and `indc` is the vector of indices of columns to extract. Omitting `indr` (respectively `indc`) means that all rows are selected (respectively all columns). Note that `indr` and `indc` can be preceded by a minus sign (-) to indicate elements not to extract.
- (b) *Extracting by logical mask*: `X[mask]`, where `mask` is a matrix of logical values TRUE/FALSE of the same size as `X`, indicating which elements to extract.

Here are a few examples of the first method:

```
> Mat <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE)
> Mat
[,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

```
> Mat[2,3]          # Extracting the element at the intersection
# of row 2 and column 3.
[1] 6
> Mat[,1]          # All rows, and only column 1.
[1] 1 4 7 10
> Mat[c(1,4),]    # All columns, and rows 1 and 4.
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 10 11 12
> Mat[3,-c(1,3)]  # Row 3 and column 2.
[1] 8
```

Here is an example with a logical mask:

```
> MatLogical <- matrix(c(TRUE,FALSE),nrow=4,ncol=3)
> MatLogical      # Is of the same size as Mat.
[,1] [,2] [,3]
[1,] TRUE TRUE TRUE
[2,] FALSE FALSE FALSE
[3,] TRUE TRUE TRUE
[4,] FALSE FALSE FALSE
> Mat[MatLogical] # Make sure that you understand this
# instruction.
[1] 1 7 2 8 3 9
```

Tip

It so happens that a matrix is stored in R as a long vector, the concatenation of all columns. Try for example the command `as.vector(Mat)`. Elements of a matrix can thus be extracted without using the form `[rows,columns]`, but instead by using vector extraction `[ind]` where `ind` is a vector of indices (or a vector of logical values) of elements to extract from the long vector.

```
> ind
[1] 2 4 6 8 3
> Mat[ind]
[1] 4 10 5 11 7
```



Warning

Using the function `"["()` sometimes leads to a change in the structure of the manipulated object. Let us see this on the example below.

```
> m <- matrix(1:6,nrow=2) ; m
   [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m[,1]
[1] 1 2
> class(m[,1])
[1] "integer"
```



It is found that the extraction has transformed our result into a simple one row vector. This can be annoying since here one could expect to get a one column matrix. But there is a solution to this problem as the following code shows:

```
> m[,1,drop=FALSE]
   [,1]
[1,]    1
[2,]    2
```

Do it yourself

Try to automatically alternate the vectors `c(0, 2, 3, 4)` and `c(1, 0, 0, 0)`, so as to get the vector `c(1, 0, 0, 2, 0, 3, 0, 4)` (hint: use the functions `cbind()`, `t()` and `as.vector()`).

As with vectors, the function `which()` can be used to return the indices of the elements of a matrix which verify some condition. For example,

```
> m <- matrix(c(1,2,3,1,2,3,2,1,3),3,3)
> m
   [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    2    1
[3,]    3    3    3
> which(m == 1)           # m is seen as the concatenation of
                           # its columns.
[1] 1 4 8
> which(m == 1,arr.ind=TRUE) # Outputs the indices as couples.
   row col
[1,]    1    1
[2,]    1    2
[3,]    2    3
```

• Insertion

Inserting elements into a matrix is done as with vectors. Elements are selected either by indices or with a logical mask and are then replaced with other elements, thanks to the affectation symbol `<-`.

```
> m
 [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    2    1
[3,]    3    3    3
> m[m!=2] <- 0
> m
 [,1] [,2] [,3]
[1,]    0    0    2
[2,]    2    2    0
[3,]    0    0    0
> Mat <- Mat[-4,] ; Mat
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> m[Mat>7] <- Mat[Mat>7]
> m
 [,1] [,2] [,3]
[1,]    0    0    2
[2,]    2    2    0
[3,]    0    8    9
```

Tip

Another function can be used to extract (and therefore also to insert) elements: `subset()`. For example, try `subset(airquality, Temp > 80, select = c(Ozone, Temp))`.



Do it yourself



```
> m1 <- matrix(c(0,22,0,23,34,0,0,0,28),ncol=3)
> m2 <- matrix(c(10,1,4,10,9,9,2,6,4),ncol=3)
> m1
 [,1] [,2] [,3]
[1,]    0    23    0
[2,]   22    34    0
[3,]    0    0    28
> m2
 [,1] [,2] [,3]
[1,]   10    10    2
[2,]    1     9    6
[3,]    4     9    4
```

Replace all non-zero values of `m1` with the corresponding values in `m2`.

Remove the second column from `m1`.

5.4.3 Extracting from/Inserting into Arrays

Extracting from and inserting into arrays is done as with matrices, except that there can be more than two dimensions. We shall therefore only list a few examples and leave it to the reader to check that they understand everything.

```
> A <- array(1:12,dim=c(2,2,3))
> A
, , 1
[,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
[,1] [,2]
[1,]    5    7
[2,]    6    8
, , 3
[,1] [,2]
[1,]    9   11
[2,]   10   12
> A[2,2,1]
[1] 4
> A[1,2,3] <- 4 # Replaces 11 with 4.
> which(A==4,arr.ind=TRUE)
      dim1 dim2 dim3
[1,]    2    2    1
[2,]    1    2    3
> A[which(A==4,arr.ind=TRUE)]
[1] 4 4
> length(A[A>4])
[1] 7
```

5.4.4 Extracting from/Inserting into Lists

- Extraction

Extracting from lists is slightly more complicated than with matrices, because each element of a list is a list itself. Using the function "["() on a list therefore outputs another list.

```
> L <- list(12,c(34,67,8),Mat,1:15,list(10,11))
> class(L)
[1] "list"
> L
[[1]]
[1] 12
[[2]]
[1] 34 67  8
```

```

[[3]]
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[[4]]
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
[[5]]
[[5]][[1]]
[1] 10
[[5]][[2]]
[1] 11
> L[2]
[[1]]
[1] 34 67 8
> class(L[2])
[1] "list"
> L[c(3,4)]
[[1]]
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[[2]]
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

```

Because lists are used to store elements of different natures, the function "[["() must be used to access the elements of a list.

```

> L[[2]]
[1] 34 67 8
> "[["(L,2)
[1] 34 67 8
> class(L[[2]])
[1] "numeric"
> L[[5]][[2]]
[1] 11

```

Warning

The following instructions generate an error:

```

> L[2,3]
Error in L[2, 3] : incorrect number of dimensions
> L[[2,3]]
Error in L[[2, 3]] : incorrect number of subscripts

```



Advanced users

R defines what is called **recursive indexing**. For example, the next example starts by retrieving the content of the second element of the list L (*i.e.* the vector `c(34, 67, 8)`) and then extracts the third element of this vector.

```
> L[[c(2,3)]] # Recursive indexing.
[1] 8
```

Furthermore, there is a mechanism in R to explicitly name the various elements of a list. The symbol \$ can then be used to extract elements by name.

```
> L <- list(cars=c("FORD", "PEUGEOT"), climate=
+                         c("Tropical", "Temperate"))
> L[["cars"]]
[1] "FORD"      "PEUGEOT"
> L$cars
[1] "FORD"      "PEUGEOT"
> L$climate
[1] "Tropical"   "Temperate"
```

- **Insertion**

Inserting elements is done as previously with the arrow `<-`.

```
> L$climate[2] <- "Continental"
> L
$cars
[1] "FORD"      "PEUGEOT"
$climate
[1] "Tropical"   "Continental"
```

Tip

A column name can include spaces. To access it, you will then need to use quotation marks.

```
> L <- list("pretty cars"=c("FORD", "PEUGEOT"))
> L
$'pretty cars'
[1] "FORD"      "PEUGEOT"
> L$"pretty cars"
[1] "FORD"      "PEUGEOT"
```

SECTION 5.5

Manipulating Character Strings

Manipulating character strings is very useful when dealing with many statistical files and when annotating graphs. We shall therefore present the major R functions in this context.

As we have seen, creating a character string is done with quotation marks "" or with the function `as.character()`.

```
> string <- c("one", "two", "three")
> string
[1] "one"    "two"    "three"
> as.character(1:3)
[1] "1" "2" "3"
```

Tip

The function `noquote()` can be used to suppress the display of quotation marks in the R output.

```
> noquote(string)
[1] one   two   three
```

The functions `sQuote()` and `dQuote()` are used for different styles of quotation marks.

The function `format()` is used to produce a personalized display, for example, data.frames.

```
> zz <- data.frame("First names"=c("Pierre", "Benoit", "Rémy"),
+                   check.names=FALSE)
> zz
  First names
1      Pierre
2      Benoit
3      Rémy
> format(zz, justify = "left")
  First names
1      Pierre
2      Benoit
3      Rémy
```



Other interesting functions to manage the display are `cat()`, `sprintf()` and `print()`.

The function `nchar()` counts the number of symbols in a string. It can be used on a vector of strings.

```
> string1 <- c("a", "ab", "B", "bba", "one", "!@", "brute")
> nchar(string1)    # Counts the number of symbols in each string.
[1] 1 2 1 3 3 2 5
> string1[nchar(string1)>2]
[1] "bba"    "one"    "brute"
```

Tip

The commands `letters` and `LETTERS` return the 26 letters of the alphabet in lower and upper case.



```
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> string1[string1 %in% c(letters, LETTERS)]
[1] "a" "B"
```

The function `paste()` is used to concatenate several strings.

```
> string2 <- c("e", "D")
> paste(string1, string2) # Concatenates the strings.
[1] "a e"      "ab D"     "B e"      "bba D"    "one e"    "!@ D"
[7] "brute e"
> paste(string1, string2, sep="-") # A separator can be included
# between the strings.
[1] "a-e"      "ab-D"     "B-e"      "bba-D"    "one-e"    "!@-D"
[7] "brute-e"
> paste(string1, string2, collapse="", sep="")
# collapse is used
# to concatenate
# the elements
# into a single
# string.
[1] "aabDBebbaDonee!@Dbrutee"
```

The function `substring()` is used to extract sub-strings from a string.

```
> substring("abcdef", first=1:3, last=2:4)
[1] "ab" "bc" "cd"
```

The function `strsplit()` is used to split a string.

```
> strsplit(c("05 Jan", "06 Feb"), split=" ")
[[1]]
[1] "05"   "Jan"
[[2]]
[1] "06"   "Feb"
```

The function `grep()` is used to search for a pattern in a vector of strings. It returns the indices of the elements of the vector containing this pattern.

```
> grep("i", c("Pierre", "Benoit", "Rems"))
[1] 1 2
```

The function `gsub()` replaces all occurrences of a pattern found in a string with another string.

```
> gsub("i", "L", c("Pierre", "Benoit", "Rems"))
[1] "PLerre" "BenoLt" "Rems"
```

The function `sub()` only replaces the first occurrence.

```
> sub("r", "L", c("Pierre", "Benoit", "Rems"))
[1] "PieLre" "BenoIt" "Rems"
```

Tip

The functions `tolower()` and `toupper()` are used, respectively, to put a string into lower or upper case.

```
> tolower("lOWER")
[1] "lower"
> toupper("UPPer")
[1] "UPPER"
```

***Do it yourself***

Input the following data.frame:

```
> X <- data.frame(date=c("03 JANUA", "02 SEPTE", "15 NOVEM"),
+                   sun=c(10,15,12))
> X
   date sun
1 03 JANUA 10
2 02 SEPTE 15
3 15 NOVEM 12
```

Remove the first column from `X` and add in two new columns: one called `day`, containing the day number coded as an numeric (1 or 2 digits), and the other called `month`, containing the month coded as four letters in lower case (hint: use the function `transform()`).

SECTION 5.6

Manipulating Dates and Time Units**5.6.1 Displaying the Current Date**

In R, there are two functions to display the current date: `Sys.time()` and `date()`.

```
> Sys.time()
[1] "2013-01-09 16:04:35 EST"
> date()
[1] "Wed Jan 9 16:04:35 2013"
```

The year, month, day, hours, minutes and seconds can be extracted as follows:

```
> as.numeric(substring(Sys.time(),c(1,6,9,12,15,18),
+                      c(4,7,10,13,16,19)))
[1] 2013     1      9     16      4     35
```

Tip

The function `system.time()` is used to find the execution time of an instruction. The function `Sys.sleep()` is used to halt the execution of a list of instructions for a given number of seconds.

5.6.2 Extracting Dates

In statistics, one often needs to extract dates from files. There are several functions in R to handle these data, which would otherwise be very difficult to manipulate.

The first useful instruction is `strptime()`, which is used to retrieve a date from a string of characters and to put it into an R object of class `POSIXlt` (named list of vectors containing informations on date and time).

```
> strptime("27/mar/73", format="%d/%b/%y")
[1] "1973-03-27"
```

In the preceding instruction, you will have noted the argument `format`, which is used to describe how the date and/or time is coded in the character string. Many codes are available; we describe them in Table 5.3 on next page.

Note

If the previous instruction outputs NA, you may need to use the following instruction to change the regional parameters (*locale*) used by default by R:

```
> Sys.setlocale("LC_TIME", "C")
```

Tip

Under Linux, the instruction `man strptime` typed into a terminal window can give you more codes.

Do it yourself

Try to read the following dates with the function `strptime()`:

```
> dates1
[1] "3jan1948"   "4jan1950"   "30apr1961"  "18sep1990"

> dates2
[1] "01/21/99 21:04:22" "03/28/99 22:19:55"
[3] "07/15/99 03:01:32"
```

Table 5.3: Codes for the function `strptime()`. The examples have been created with `format(Sys.time(), "%letter")`

Code	Description	Example
%a	Day of the week, abbreviated	Mon
%A	Day of the week, full name	Monday
%b	Month, abbreviated	Dec
%B	Month, full name	December
%c	Date and time, locale-specific	Mon 20 Dec 2010 13:06:24 CEST
%d	Day of the month (01–31)	20
%H	Hours (0–24)	13
%I	Hours (0–12)	01
%j	Day of the year (001–366)	354
%m	Month (01–12)	12
%M	Minutes (00–59)	06
%S	Seconds (00–61), with 2 “leap seconds”	24
%U	Week of the year (00–53); the 1 st Sunday is counted as day 1 of week 1	51
%w	Day of the week (0–6); Sunday is 0	1
%W	Week of the year (00–53); the 1 st Monday is counted as day 1 of week 1	51
%x	Date, locale-specific	2010-12-20
%X	Time, locale-specific	13:06:24
%y	Year without the century	10
%Y	Year with the century	2010
%z	Time offset from Greenwich; ‘-0800’ is 8 h West of Greenwich	+0100
%Z	Time zone as a character string (output only)	CEST

Note that the functions `weekdays()` and `months()` can be used to retrieve the day and month of a date in the `POSIXlt` format.

5.6.3 Operations on Dates

Before manipulating dates, you should always start by converting dates and times into objects of the class `POSIXlt` or `POSIXct`. There are two functions in `R` to this effect: `as.POSIXct()`, which gives the number of seconds elapsed since 1st January 1970 as a numeric vector, and `as.POSIXlt()`, which is a list of vectors representing

`sec` 0–61: seconds

`min` 0–59: minutes

`hour` 0–23: hours

`mday` 1–31: day of the month

`mon` 0–11: number of months since the 1st month of the year

`year` : number of years since 1900

wday 0–6: day of the week, starting on Sunday
yday 0–365: day of the year
isdst : Daylight Saving Time (DST) flag. Positive if DST is observed, zero otherwise (negative if unknown)

Here are a few instructions using these functions:

```
> z <- Sys.time() # In the POSIXct format.
> class(z) ; is.double(z)
[1] "POSIXt" "POSIXct"
[1] TRUE
> z
[1] "2013-01-09 16:04:35 EST"
> as.numeric(z) # Number of seconds since 1st January 1970.
[1] 1357765476
> # The origin can be changed:
> as.POSIXct(as.numeric(z), origin="1960-01-01")
[1] "2003-01-09 21:04:35 EST"
> # About fourty years have elapsed:
> as.numeric(z)/60/60/24/7/c(53,52)
[1] 42.35816 43.17274
> z <- as.POSIXlt(z) # In the POSIXlt format.
> class(z) ; is.list(z)
[1] "POSIXt" "POSIXlt"
[1] TRUE
> z
[1] "2013-01-09 16:04:35 EST"

> names(z)
[1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"
[8] "yday"   "isdst"
> z$year # Number of years since 1900.
[1] 113
```

Note that the functions `as.POSIXct()` and `as.POSIXlt()` can be used either on vectors of numeric values or on vectors of character strings. In the former case, you will need to give a value to the argument `origin`, as a character string representing a date. In the latter case, each character string must be in a format such as "2001-02-03" or "2001/02/03", optionally followed by a space and a time in the format "14:52" or "14:52:03". It might be useful to use the function `strptime()` to get a format compatible with these functions (see Table 5.3 for a description).

```
> as.POSIXct("2001/02/03")
[1] "2001-02-03 EST"
> as.numeric(as.POSIXct("2001/02/03"))
[1] 981176400
> as.POSIXlt("2001/02/03")$wday
[1] 6
> lct <- Sys.getlocale("LC_TIME")
> Sys.setlocale("LC_TIME", "C") # See Note on page 112.
[1] "C"
> as.POSIXlt(strptime("27/mar/73", format="%d/%b/%y"))
[1] "1973-03-27"
> Sys.setlocale("LC_TIME", lct) # To recover the locale.
[1] "fr_FR.UTF-8"
```

Note

The class `Date` can also be used to represent dates.

```
> z <- as.Date(c("2006-06-01", "2007-01-01"))
> class(z)
[1] "Date"
> z[1] + 100 # Add 100 days.
[1] "2006-09-09"
> z[2]-z[1]
Time difference of 214 days
> z[2] < z[1]
[1] FALSE
```



The advantage of storing dates in objects with one of the classes described above, apart from the pretty display, is that operations can then be made on these dates (difference between two dates, anteriority test,...), as illustrated in the following examples:

```
> date2 <- as.POSIXlt("2009-04-15")
> date1 <- as.POSIXlt("2000-11-24")
> date2-date1
Time difference of 3063.958 days
> difftime(date2,date1,units="hours")
Time difference of 73535 hours
> date1 <= date2
[1] TRUE
```

Advanced users

The package `chron` includes many functionalities to handle dates.



SECTION 5.7

Control Flow

Like all programming languages, **R** includes the necessary control structures to direct the execution flow of a program.

5.7.1 Conditional Instructions

- **Instruction switch()**

It is used in the following way:

```
switch(<expr:test>, <expr:case1>=<code1>, <expr:case2>=<code2>,
etc.)
```

In the instruction above, `<expr:test>` is either a number or a character string. This instruction outputs `<code1>` if `<expr:test>` is equal to `<expr:case1>`, `<code2>` if `<expr:test>` is equal to `<expr:case2>` and so on. If `<expr:test>` is not equal to any of the `<expr:case>`, the function `switch()` outputs NULL.

Here is an example:

```
> x <- rcauchy(10) # Generate ten random numbers from a Cauchy
# distribution.
> my.input <- "mean"
> switch(my.input, mean = mean(x), median = median(x))
[1] -0.5472605
> my.input <- "median"
> switch(my.input, mean = mean(x), median = median(x))
[1] -0.3508165
> my.input <- "variance"
> switch(my.input, mean = mean(x), median = median(x))
```

Note

You can also include a single unnamed value, *i.e.* a `<codei>` without the associated `<expr:casei>=`. This value will then be output instead of NULL when the value of the argument EXPR is not equal to any of the cases.

```
> switch(EXPR = "b", a=4, b=2:3, "Else: nothing")
[1] 2 3
> switch(EXPR = "QQ", a=4, b=2:3, "Else: nothing")
[1] "Else: nothing"
```

- **Instructions if() and else**

The conditional instruction `if()` can be used in the two following ways:

```
if (<cond>) <expr:true>
or
```

```
if (<cond>) <expr:true> else <expr:false>
```

The argument `<cond>` must be a logical value, which therefore takes one of the values `TRUE` or `FALSE`. Note that `<cond>` is first transformed by `as.logical(<cond>)`, so that real numbers are allowed (`0` is the only number to be transformed into `FALSE`) as well as character strings "T" or "TRUE" and "F"

or "FALSE". Also note that <cond> must be of length 1. Otherwise, only the first element of <cond> will be taken into account, and R will display a warning.

In practice, of course, <cond> is often the result of an elaborate logical operation, computed with the logical operators we described above. Here is an example of how these instructions can be used. Make sure you understand them well.

```
> if (TRUE) 1+1
[1] 2
> x <- 2
> y <- -3
> if (x <= y) {
+   z <- y-x
+   print("x smaller than y")
+ } else {
+   z <- x-y
+   print("x larger than y")
+ }
[1] "x larger than y"
[1] 5
```

Tip

The function `ifelse()` is used to execute one or the other of two functions applied to a vector, depending on the values of a logical condition. For example,

```
> x <- c(3:-2)
> sqrt(ifelse(x >= 0, x, NA))
[1] 1.732051 1.414214 1.000000 0.000000      NA          NA
```



- **Preferred logical operators**

Make sure you use logical operators well. For conditional instructions, we advise you to use:

- `x && y` rather than `x & y`
- `x | | y` rather than `x | y`

This next example shows why. If you use the long form `&&`, the logical conditions after the `if` are evaluated from left to right, until a `FALSE` is found.

```
> as.logical(x <- 2) # as.logical(x, non-zero real number)
                      # outputs TRUE.
[1] TRUE
> x
[1] 2
> rm(x) # Remove x.
> if (FALSE & as.logical(x <- 2)) 4^7 # <cond> is evaluated to
                                         # be FALSE. Both parts
                                         # of <cond> are
                                         # evaluated.
```

```
> x
[1] 2
> if (FALSE && (x <- 3)) 4*7 # If you use &&, only the first
                           # part of <cond> is evaluated.
> x
[1] 2
```

When you use the long form `||`, the logical conditions after `if` are evaluated from left to right until a TRUE is found. You should therefore use the long form, since it will make your code run faster.

- **The function `all.equal()` for the `if()` instruction**

When using the `if()` instruction (above all for real values), you should:

- use `isTRUE(all.equal(x,y))` rather than `x == y`
- use `!isTRUE(all.equal(x,y))` rather than `x != y`

This point is illustrated in the following example, in which `x` and `y` are not exactly equal, because of machine precision:

```
> x <- 0.1
> y <- 0.1
> x==y
[1] TRUE
> x <- 0.2-0.1    # It seems that
> y <- 0.3-0.2    # x is equal to y.
> x == y          # This is not the case, because the computer
                   # has a limited precision. See Sect. 5.9.
[1] FALSE
> all.equal(x,y,tolerance=10^-6) # The function all.equal()
                                # solves this issue.
[1] TRUE
```

Tip

The function `all.equal()` takes an optional argument `tolerance`, used to fix the tolerance limit below which the difference between two values is taken to be zero.

5.7.2 Loop Instructions

A loop is a control structure which allows a portion of code to be executed several times in a row, until an exit condition is satisfied or until a pre-specified number of loops has been reached.

There are three loop instructions in R: `for`, `while()` and `repeat`. In addition, the reserved words `next` and `break` give extra control on code execution. The instruction `break` immediately exits the current loop. The instruction `next` takes the program execution back to the beginning of the loop, so that the next iteration

(if it exists) of the loop is then executed. For the current iteration, no instruction after `next` is executed.

• Instruction `for`

The syntax for this instruction is as follows:

```
for (i in vect) <Instructions>
```

Here are two examples:

```
> for (i in 1:3) print(i)
[1] 1
[1] 2
[1] 3
> x <- c(1,3,7,2)
> for (var in x) print(2*var)
[1] 2
[1] 6
[1] 14
[1] 4
```

Tip

The following list of instructions prints a decreasing counter:
`n<-100;`
`for (i in 1:n) {flush.console();cat(n-i,"\\r");Sys.sleep(0.1)}`



• Instruction `while()`

The syntax for this instruction is as follows:

```
while(<condition>) <expression>
```

For example,

```
> x <- 2
> y <- 1
> while(x+y<7) x <- x+y
> x
[1] 6
```

• Instructions `next` and `break`

```
> for (i in 1:4) {
+   if (i == 3) break
+   for (j in 6:8) {
+     if (j==7) next
```

```
+   j <- i+j
+
+ }
> i
[1] 3
> j
[1] 10
```

- Instructions **repeat** and **break**

```
> i <- 0
> repeat {
+   i<-i+1
+   if (i==4) break
+ }
```

Warning

Whenever possible, it is better to avoid loops in R, since they often slow down the execution (as measured by the function `system.time()`). Indeed, most operations in R are vectorized, which means that they can operate on vectors, and these calculations are done in a compiled language, which is much faster:



```
> system.time(for (i in 1:1000000) sqrt(i))
  user  system elapsed
  0.342  0.001  0.343
> system.time(sqrt(1:1000000))
  user  system elapsed
  0.018  0.004  0.022
```

Moreover, functions such as `apply()`, `tapply()` and `sapply()` give a way to use loops in an implicit, and often very useful, manner.

— SECTION 5.8 —

Creating Functions

We saw in Sect. 3.1.4, page 43, some brief notions on executing functions in R. The R language can also be used to create your own functions. We propose an overview in this section. The reader should linger on all the code given in this section, to ensure it is well understood.

See also

A more formal presentation of function writing is given in Chap. 8.

To illustrate simply the function creation process, we shall focus on the computation of the body mass index (BMI), from the weight (in kg) and the height (in m), using the well-known formula

$$BMI = \frac{Weight}{Height^2}.$$

This is easily programmed in R as follows:

```
> BMI <- function(weight,height) {
+   bmi <- weight/height^2
+   names(bmi) <- "BMI"
+   return(bmi)
+ }
```

Warning

The function `return()` is optional in the code above, but you should take the habit of using it. Indeed, there are contexts where it is essential:

```
> f <- function(x) {
+   res <- vector("numeric",length(x))
+   for (i in 1:10) {
+     res[i] <- rnorm(1) + x[i]
+   }
+ # Forgot to include return(res)
+ }
> f(1:10) # Does not output anything!
```



We can now execute the function `BMI()` we just created:

```
> BMI(70,1.82)
      BMI
21.13271
> BMI(1.82,70) # Note that it is not possible to swap the
                  # arguments of a function,
      BMI
0.0003714286
> BMI(height=1.82,weight=70) # unless they are preceded by their
                            # names.
      BMI
21.13271
```

This function only outputs a single value. The code below outputs a list of several variables.

```
> BMI <- function(weight,height) {
+   bmi <- weight/height^2
+   res <- list(weight,height,bmi)
+   return(res)
+ }
```

The next instruction shows that the new function `BMI()` returns a list of unnamed elements.

```
> BMI(70,1.82)
[[1]]
[1] 70
[[2]]
[1] 1.82
[[3]]
[1] 21.13271
```

Do it yourself



Write a function called `biroot()` which calculates the roots of a polynomial of order 2, *i.e.* the values x which solve the equation $ax^2 + bx + c = 0$. Recall that they can be complex and are given by

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

with

$$\Delta = b^2 - 4ac.$$

If $\Delta = 0$, there is only one solution; return only that value. If $\Delta < 0$, then the square root of Δ is the complex number z such that $z^2 = \Delta$ (hint: use the instruction `as.complex(Delta)`).

Compare the results of your function with those given by the function `polyroot()` (remember to use the online help for this function: `?polyroot`).

To name the elements of the list, you can use the following code:

```
> BMI <- function(weight,height) {
+   bmi <- weight/height^2
+   res <- list(Weight=weight,Height=height,BMI=bmi)
+   return(res)
+ }
```

which gives the following result:

```
> BMI(70,1.82)
$Weight
[1] 70
$Height
[1] 1.82
$BMI
[1] 21.13271
```

Now assume we wish to calculate the BMI of several individuals, for example, John and Peter:

```
> John <- c(74,1.90)
> Peter <- c(70,1.82)
> mydata <- rbind(John,Peter)
```

We might use the following code:

```
> for (i in 1:2) {
+ print(BMI(mydata[i,1],mydata[i,2]))
+
$Weight
John
74
$Height
John
1.9
$BMI
John
20.49861
$Weight
Peter
70
$Height
Peter
1.82
$BMI
Peter
21.13271
```

But the attentive reader will have noticed that many R functions (including the division, multiplication and exponentiation operators) work very well with vectors. The following output is thus not surprising:

```
> BMI(c(70,74),c(1.82,1.90))
$Weight
[1] 70 74
$Height
[1] 1.82 1.90
$BMI
[1] 21.13271 20.49861
```

The following code illustrates the use of an argument with a default value, as well as the function `stop()` which can handle some input errors.

```
> BMI <- function(weight,height,height.unit="m") {
+ if (length(weight) != length(height))
+ stop("The vectors weight and height must have the same length.")
+ if (height.unit == "cm") height <- height/100
+ bmi <- weight/height^2
+ res <- list(Weight=weight,Height=height,BMI=bmi)
+ return(res)
+ }
```

Advanced users

Using the function `stop()` can lead to annoyances. For example, in a simulation study, one often has to call a function repeatedly. If, in one of the calls, the function returns an error, the simulation is stopped. It is advisable to use the function `try()`. If the function encounters an error, the error message is stored in an object without stopping the simulation.



```
> set.seed(123)
> x <- rnorm(50)
> doit <- function(x) {
+   x <- sample(x, replace=TRUE)
+   if(length(unique(x)) > 30) mean(x)
+   else {stop("too few unique points")}
+ }
> res <- lapply(1:100, function(i) try(doit(x), TRUE))

> res[8:10]
[[1]]
[1] 0.2030573
[[2]]
[1] "Error in doit(x) : too few unique points"
attr(,"class")
[1] "try-error"
[[3]]
[1] 0.235046
```

It is possible to classify individuals into weight categories depending on their BMI. The correspondence between BMI and categories is given in Table 5.4.

The next function outputs a user's weight category given their BMI.

```
> weight.category <- function(bmi) {
+   if (bmi<16.5) category <- "severely underweight" else {
+     if (bmi<18.5) category <- "underweight" else {
+       if (bmi<25) category <- "normal weight" else {
+         if (bmi<30) category <- "overweight" else {
+           if (bmi<35) category <- "moderate obesity" else {
+             if (bmi<40) category <- "severe obesity" else {
+               category <- "morbid obesity"}}}}}}
+   cat(paste("Your BMI is: ",round(bmi,3),".\n"))
+ This corresponds to category: ",category,".\n",sep=""))
+ }
```

Here is an example of use:

Table 5.4: Correspondence between BMI and weight categories

Severely underweight	Underweight	Normal weight	Overweight	Moderate obesity	Severe obesity	Morbid obesity
[15; 16.5]	[16.5; 18.5]	[18.5; 25]	[25; 30]	[30; 35]	[35; 40]	[40; 41]

```
> weight.category(BMI(70,1.82)$BMI)
Your BMI is: 21.133.
This corresponds to category: normal weight.
```

The code of the function `weight.category()` can be simplified with the function `switch()`, as can be seen below:

```
> weight.category <- function(bmi) {
+   intervals.BMI <- c(15,16.5,18.5,25,30,35,40,41)
+   code <- as.character(rank(c(bmi,intervals.BMI),
+     ties.method="max"))[1]
+   category <- switch(code,"2"="severely underweight",
+     "3"="underweight","4"="normal weight","5"="overweight","6"=
+     "moderate obesity","7"="severe obesity","8"="morbid obesity")
+   cat(paste("Your BMI is: ",round(bmi,3),".\n"))
+   This corresponds to weight category: ,category,".\n",sep=""))
+ }

> weight.category(BMI(70,1.82)$BMI)
Your BMI is: 21.133.
This corresponds to weight category: normal weight.
```

However, this function outputs wrong results when used on a vector:

```
> weight.category(BMI(c(70,74),c(1.82,1.90))$BMI)
Your BMI is: 21.133.
This corresponds to weight category: overweight.
Your BMI is: 20.499.
This corresponds to weight category: overweight.
```

Our function can be improved so that it works on several individuals simultaneously. Note the use of the reserved word `NULL` and of the function `is.null()` which handles shrewdly the parameter `names`.

```
> weight.category <- function(bmi,names=NULL) {
+   intervals.BMI <- c(15,16.5,18.5,25,30,35,40,41)
+   n <- length(bmi)
+   if (is.null(names)) names <- paste("Subject number",1:n)
+   if (length(names) != n) stop(paste("The vector of 'names'",
+     "must be of length",n))
+   code <- vector("integer",length=n)
+   category <- vector("character",length=n)
+   for (i in 1:n) {
+     code[i] <- as.character(rank(c(bmi[i],intervals.BMI),
+       ties.method="max"))[1]
+     category[i] <- switch(code[i],"2"="severely underweight",
+       "3"="underweight","4"="normal weight","5"="overweight","6"=
+       "moderate obesity","7"="severe obesity","8"="morbid obesity")
+     cat(paste(names[i],":\nYour BMI is: ",round(bmi[i],3),".\n"))
+   This corresponds to weight category: ,category[i],".\n",sep=""))
+ }
+ }
```

Here is an example of use.

```
> weight.category(BMI(c(70,74),c(1.82,1.90))$BMI)
Subject number 1:
Your BMI is: 21.133.
This corresponds to weight category: normal weight.
Subject number 2:
Your BMI is: 20.499.
This corresponds to weight category: normal weight.
```

Tip

In the previous function, we declared, directly with the correct length, the vectors `code` and `corpulence`, thanks to the function `vector()`. However, it is also possible to handle a vector without a predefined dimension (ever because we do not know its length initially or for another reason). This is illustrated on the next example, where we compute the first terms of the Fibonacci sequence, which is defined by

$$a_1 = 0, a_2 = 1, a_i = a_{i-1} + a_{i-2}, \forall i \geq 3.$$

```
> a <- c(0, 1)
> for (i in 3:10) a <- c(a, a[i-1]+a[i-2])
> a
[1] 0 1 1 2 3 5 8 13 21 34
```

We could also use the function `while()` to display all the terms of the Fibonacci sequence until the first term is greater than 1000.



```
> a <- c(0,1)
> i <- 3
> while(a[i-1]<1000) {
+   a <- c(a, a[i-1]+a[i-2])
+   i <- i+1
+ }
> a
[1] 0 1 1 2 3 5 8 13 21 34 55
[12] 89 144 233 377 610 987 1597
```

We could also have used the instruction `break` for a slightly different function.

```
> a <- c(0,1)
> i <- 3
> while(TRUE) { # Create un infinite loop.
+   a <- c(a, a[i-1]+a[i-2])
+   if (a[i]>1000) break; # Stops the loop.
+   i <- i+1
+ }
> a
[1] 0 1 1 2 3 5 8 13 21 34 55
[12] 89 144 233 377 610 987 1597
```

Warning

The programming method presented in the previous Tip is not necessarily optimal in terms of memory allocation, as we shall see in Chap. 8.



SECTION 5.9

† Fixed-Point and Floating Point Number Representation

This (rather technical) note aims to help the R user to identify and prevent some mistakes due to use of the so-called *floating point* numbers.

5.9.1 Representing a Number in a Base

Given a *base* b (integer value, greater than or equal to 2), any real number $x \in \mathbb{R}$ can be written in *fixed-point* representation as follows:

$$x = \sum_{i=-\infty}^{i=+\infty} m_i b^i,$$

where the coefficients m_i (also called the digits) belong to the set $\{0, 1, \dots, b - 1\}$, $\forall i \in \mathbb{Z}$.

For example, the number $x = 10.625$ can be written in *decimal notation* ($b = 10$) as

$$x = 1 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3},$$

hence the notation **10.625** which gives the coefficients of the representation (or *digits*). By noting that $10.625 = 8 + 2 + 0.5 + 0.125$, this same number can be written in *binary notation* ($b = 2$; hence the only digits are 0 and 1) as

$$x = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}.$$

We thus have the representation of the number 10.625 in binary form **1010.101**, which makes explicit the coefficients of the binary representation.

Tip

The package associated with this book includes the functions `dec2bin()` and `bin2dec()` which can be used to switch from decimal to binary representation and vice versa:



```
> bin2dec(1010.101)
[1] 10.625
> dec2bin(10.625,3)
[1] "1010.101"
```

Note


A number with finite base 10 expansion (finite number of digits after the decimal separator) is called a *decimal number*. A number with finite base 2 expansion is called a *dyadic number*. Obviously, some numbers are neither decimal nor dyadic. Note that a dyadic number is always also a decimal number, but the converse is false. Thus a loss of information is always possible when switching from one representation to another. Furthermore, dyadic decimal fractions have the same number of digits as their binary equivalents whereas non-dyadic decimal values have infinite binary equivalents.

A computer is a machine built to work with binary numbers, because the two digits 0 and 1 are easily translated as the presence or absence of an electric current. Because fixed-point representations are often very costly in *bits* (a bit is a binary digit, *i.e.* 0 or 1), computers generally use floating point representations. These are described in the next subsection.

5.9.2 Floating Point Representations

5.9.2.1 Definitions

Given a *base* b (*e.g.*, $b = 2$ for the binary system or $b = 10$ for the decimal system, to cite only the two most classical systems), any real number $x \in \mathbb{R}$ can be written as

$$x = (-1)^s m b^e$$

where

- s is called the sign bit (of x) and is worth 0 or 1;
- m is the *significand* or *mantissa* and can be written $m = m_1.m_2m_3\cdots m_\infty$ where each $m_i \in \{0, 1, \dots, b - 1\}$ is called a digit;
- $e \in \mathbb{Z}$ is called the *exponent*.

This notation is called the *floating point representation* of x in base b . The integer m_1 is the *integer part* of the significand and the other digits $m_2 \cdots m_\infty$ are the *fractional part* of the significand. Note that we must impose a constraint that the first digit be non-zero ($m_1 \neq 0$), so as to insure that the representation is unique. Of course, this constraint cannot be applied for the particular case $x = 0$.

Let us give a first example. In the decimal system, the number -0.6345 can be written in floating point representation by taking $s = 1$, $m_1 = 6$, $m_2 = 3$, $m_3 = 4$, $m_4 = 5$, $m_i = 0$, $\forall i > 4$, $b = 10$ and $e = -1$:

$$-0.6345 = (-1)^1 6.345 \times 10^{-1}.$$

Note

This explains the name “floating point”: the decimal point has been moved, to express the same number in a different way.



Actually, in a computer, any real number x is represented in a binary base, with bits (0 or 1), using the slightly different formula:

$$x = (-1)^s(1 + f)2^{(e-1023)} \quad (5.1)$$

where s is an integer coded on a single bit (called sign bit), e is a non-negative integer coded on 11 bits (thus taking the values from 0 to $2^{11} - 1 = 2047$) and $f \in [0; 1)$ is a number coded on 52 bits, thus written

$$f = \sum_{i=1}^{i=52} k_i 2^{-i}, \quad (5.2)$$

where the k_i 's take the values 0 or 1. By convention, the value $e = 0$ means that $x = 0$, and the value $e = 2047$ means that $x = +\text{Inf}$ or $x = -\text{Inf}$, depending on the value on s . The following two examples illustrate this:

```
> s <- 1; e <- 2047; f <- 0; x <- (-1)^s * 2^(e-1023) * (1+f); x
[1] -Inf
> s <- 0; e <- 2047; f <- 0; x <- (-1)^s * 2^(e-1023) * (1+f); x
[1] Inf
```

5.9.2.2 Limitations of This Representation due to the Significand

The vast majority of modern computers use the floating point representation. However, it is essential to note that because of physical limitations, this representation is not perfect. Indeed, the significand m can only have a limited (finite) number of digits, and the relative integer e is bounded, $e \in [e_{\min}, e_{\max}]$, since it is not possible to code an infinite number of integers on a physical machine made of a finite number of components.

Here is a simple example, which should help understand this limitation. The floating point representation in base 10 of the real number $1/3$ is written:

$$1/3 = (-1)^0 \times 3.33333\cdots 333\cdots \times 10^{-1}.$$

It is of course not possible to store an infinite number of 3s on a computer. A more striking example is the number $\pi = 3.14159265358979\dots$ for which there is no pattern in the digits after the decimal point. But it is also true for numbers which appear simpler, such as 0.1 or 0.9.

This is illustrated in the output below, for the number 0.9, which can be written in floating point representation as $0.9 = (-1)^0 \times 2^{-1} \times (1 + f)$ with $f = 0.8$. However, since f is coded on “only” 52 bits (see (5.2)), the computer only keeps an approximation. The display function `formatC()` gives an idea of the value used by R.

We can now display the value stored and used by the computer instead of the value 0.9:

```
> formatc(0.9,50)
[1] "0.90000000000000002220446049250313080847263336181641"
> formatc((-1)^0 * 2^(-1) * (1+f),50)
[1] "0.90000000000000002220446049250313080847263336181641"
> formatc((-1)^0 * 2^(-1) * (1+0.8),50)
[1] "0.90000000000000002220446049250313080847263336181641"
```

5.9.2.3 Avoiding Some Numerical Pitfalls

First, we shed light on a numerical oddity.

```
> identical(1.0-0.9,0.1)
[1] FALSE
> (1.0-0.9) == 0.1
[1] FALSE
```

We can now understand this output. Indeed, the numbers $1.0 - 0.9$ and 0.1 are represented in different ways, because even though 1 is perfectly represented, 0.1 and 0.9 are not.

```
> formatC(1.0,50)
[1] "
> formatC(0.9,50)
[1] "0.9000000000000002220446049250313080847263336181641"
> formatC(1.0-0.9,50)
[1] "0.09999999999999977795539507496869191527366638183594"
> formatC(0.1,50)
[1] "0.100000000000000055511151231257827021181583404541"
```

and we thus have

```
> formatC(1.0 -
+ 0.9000000000000002220446049250313080847263336181641,50)
[1] "0.09999999999999977795539507496869191527366638183594"
> # which is different from:
> formatC(0.1,50)
[1] "0.100000000000000055511151231257827021181583404541"
```

Tip

It is better to use the function `all.equal()` to compare two numbers, because it includes a numerical tolerance argument.



```
> all.equal(1.0-0.9,0.1,tol=10^(-6))
[1] TRUE
```

Warning

Do not use constructions such as `while(x != 0)` or `if (x != 0)`. Indeed, if `x` happens to take a value such as $1.0 - 0.9 - 0.1$ when your code runs, those two instructions will not behave as expected. You should use instead `isTRUE(all.equal(x,0))` and `isTRUE(all.equal(x,0))`.



We leave it to the reader to understand the output below.

```
> as.integer(100*(1-.34))
[1] 65
> floor(100*(1-.34))
[1] 65
> round(100*(1-.34))
[1] 66
> 100*(1-.34)-66
[1] -1.421085e-14
> floor(100*(1-.38))
[1] 62
> round(100*(1-.38))
[1] 62
> 100*(1-.38)-62
[1] 0
```

See also

You may find [7] useful in this context.

5.9.2.4 Limitations of This Representation due to the Exponent

- Smallest and largest numbers which can be represented

Another problem due to the floating point representation is as follows. Since the exponent e of the representation

$$x = (-1)^s(1 + f)2^{(e-1023)} \quad (5.3)$$

is necessarily bounded (since it is coded on 11 bits), there exist a smallest and largest real number which can be represented on a given computer. Trying to represent a number out of this range should lead to an underflow or overflow. R is rather well conceived and will return the value `-Inf` or `+Inf`. The following R commands illustrate this point:

```
> .Machine$double.xmin # Smallest real number which can be coded:
[1] 2.225074e-308
> # which can also be found by:
> s <- 0; e <- 0; f <- sum(2^(-(1:52)))
> x <- (-1)^s * 2^(e-1023) * (1+f); x
[1] 2.225074e-308
> .Machine$double.xmax # Largest real number which can be coded:
[1] 1.797693e+308
> # which can also be found by:
> s <- 0; e <- 2046; f <- sum(2^(-(1:52)))
> x=(-1)^s * 2^(e-1023) * (1+f); x
[1] 1.797693e+308
```

Tip

The package `Brobdingnag` allows handling much larger numbers.

- Gaps between two numbers

One last warning may be useful: it is important to note that it is not possible to get some numbers between `.Machine$double.xmin` and `.Machine$double.xmax` (even integers).

For example, take the number 2^{150} , which is represented as

$$(-1)^0(1 + 0)2^{(1173 - 1023)}.$$

The next number that the computer can code is given by the values $s = 0$, $e = 1173$ and $f = 2^{-52}$ (smallest non-zero fractional part of the significand), i.e.

$$(-1)^0(1 + 2^{-52})2^{150} = 2^{150} + 2^{150-52} = 2^{150} + 2^{98}.$$

There can thus be a huge “gap” (of length $2^{98} \approx 3.2\text{e+}29$ here) between two “successive” numbers!

This explains the following oddity:

```
> a <- 2^150; b <- a + 2^97; b == a
[1] TRUE
> a <- 2^150; b <- a + 2^98; b == a
[1] FALSE
```

Tip

More information on the computer’s limitations in representing numbers are given by the instruction `.Machine`:

```
> noquote(unlist(format(.Machine)))
      double.eps          double.neg.eps
      2.220446e-16        1.110223e-16
      double.xmin          double.xmax
      2.225074e-308       1.797693e+308
      double.base          double.digits
                      2           53
      double.rounding       double.guard
                      5           0
      double.ulp.digits    double.neg.ulp.digits
                      -52          -53
      double.exponent       double.min.exp
                      11           -1022
      double.max.exp       integer.max
                      1024          2147483647
      sizeof.long            sizeof.longlong
                      8             8
      sizeof.longdouble     sizeof.pointer
                      16            8
```



See also

We refer the interested reader to the document *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, available at the URL:
<http://biostatisticien.eu/springer/FloatingPoint.pdf>.



Memorandum

length(): length of a vector
sort(): sort the elements of a vector
rev(): rearrange the elements of a vector in reverse order
order(): return the vector of order ranks of the elements of its effective argument
unique(): remove the duplicates from a vector
dim(): size of a matrix or data.frame
nrow(), **ncol()**: number of rows and columns
dimnames(): names of rows and columns
rownames(), **colnames()**: names of rows and columns
rbind(), **cbind()**: merge rows and columns of a matrix
merge(): smart merge of columns
apply(): apply a function to the rows or columns of a matrix
lapply(), **sapply()**: apply a function to the elements of a list
`<, <=, >, >=, ==, !=`: comparison logical operators
`!, &, &&, |, ||`: term-wise logical operators
any(x): return TRUE if one of the x_i is true
all(x): return TRUE if all the x_i are true
if(), **else**, **switch**: conditional instructions
for, **while()**, **repeat**: loop instructions
"["): extraction operator for vectors and matrices
"[["): extraction operator for lists
which(): indices of the values TRUE of a logical object
nchar(): number of characters in a string
paste(): concatenate two strings
substring(): extract sub-strings
strsplit(): split strings
grep(): search for a pattern in a string
sub(), **gsub()**: replace occurrences of a pattern in a string
Sys.time(): display the date
strptime(): extract dates from a string
as.POSIXlt(): convert to the POSIXlt format
difftime(): calculate the difference between two dates



Exercises

- 5.1-** What is the output of this instruction: `c(1,4)*c(2,3)`?
- 5.2-** What is the output of this instruction: `matrix(1:2, ncol=2, nrow=2)`?
- 5.3-** How can you retrieve the names of rows and columns of a data.frame?
- 5.4-** Give the instruction to merge these two tables:

```

> X
      Gender Weight
Jack       M     80
Julia      F     60
> Y
      Eyes Height
  
```

```
Jack    Blue     180
Julia   Green    160
```

- 5.5- Give the instruction to calculate the product of all the elements (respectively, of all the elements of each column) of a numerical matrix X.
- 5.6- What is the output of this instruction: `vec<-c(2,4,6,8,3); vec[2];vec[-2]`?
- 5.7- The height and weight of several men were measured. The measurements are stored in the vectors `weight` and `height`. Give the R instruction to get the weight of the men whose height is greater than 180 cm.
- 5.8- What is the output of this instruction:
`Mat<-matrix(1:12,nrow=4,byrow=TRUE);Mat[3,];Mat[2,2:3]`?
- 5.9- How could you replace the fourth component of the following list with `1:10`?
`L<-list(12,c(34,67),Mat,1:15,list(10,11))`
- 5.10- What is the output of this instruction: `L[[2]][2]`?
- 5.11- Give the R instruction which outputs the weights and heights of all women in the following table (you can use the function `attach()`):

```
> X
  weight height gender
1      79     163     M
2      90     163     F
3      87     198     M
4      63     164     F
5      90     168     F
6      71     178     F
7      58     191     M
8      80     194     F
9      91     185     F
10     89     176     M
```

- 5.12- What is the output of this instruction: `(1:3)[any(c(T,F,T))]`? And this one: `(1:3)[all(c(T,F,T))]`?
- 5.13- What is the output of this instruction: `c(T,T,F) | c(F,T,F)`? And this one: `c(T,T,F) || c(F,T,F)`?
- 5.14- What is the output of this instruction: `nchar(c("abcd","efgh"))`?
- 5.15- What is the output of this instruction:
`paste(c("a","b"),c("c","d"),collapse="",sep="")`?
- 5.16- What is the output of this instruction: `strsplit(c("ab;cd"),";")`?
- 5.17- What is the output of this instruction: `substring("abcdef",3,c(2,4))`?
- 5.18- How could you transform the upper case into lower case in the following vector?
`c("Jack","Julia","William")`
- 5.19- Which function is used to retrieve a date from a character string?
- 5.20- Can you explain why the second number of the last output is not equal to 36.21313?

```
> logp <- function(x) log(max(x,exp(1)))
> x <- -2.4
> delta <- 0.1
```

```

> (abs(x))^(4+delta)/(logp(abs(x)))^2
[1] 36.21313
> x <- seq(from=-2.8,to=-2,length=3)
> x
[1] -2.8 -2.4 -2.0
> (abs(x))^(4+delta)/(logp(abs(x)))^2
[1] 64.26795 34.15959 16.17594

```

Propose a solution to this problem.



Worksheet

Manipulating Various Data Sets

A- Manipulating a Few Data Sets Presented at the Beginning of the Book

These files can be downloaded from the URL <http://www.biostatisticien.eu/springeR/>. Note that you can also append the file name at the end of this URL to download the file directly from R (e.g., <http://www.biostatisticien.eu/springeR/nutrien1.xls>):

- *Data set NutriElderly:*

The data file `nutrition_elderly.xls`, described earlier in this book, is in fact the merge of two initial files, entered by different operators. We propose to reconstruct the file for the following cases. You will only use R and will not edit the file by hand.

- 5.1-** The individuals are initially listed separately in two files (`nutrien1.xls` and `nutrien2.xls`). Note that the variable names are in upper case in the first file and in lower case in the second.
- 5.2-** Some individuals are listed in both files (`nutrien3.xls` and `nutrien4.xls`). The variable names are identical.
- 5.3-** Same question as 5.2, but errors have slipped in and you will need to detect the corresponding individuals, for example, those with a weight greater than 200 kg (`nutrien5.xls` and `nutrien6.xls`).
- 5.4-** The variables are split between two files (`nutrien7.xls` and `nutrien8.xls`), which contain the same individuals.
- 5.5-** Same question as 5.4, but for one variable, too many values are missing. Remove that variable (`nutrien9.xls` and `nutrien10.xls`).
- 5.6-** Same question as 5.4, but for one individual, too many values are missing. Remove that individual (`nutrien11.xls` and `nutrien12.xls`).
- 5.7-** In the file `nutrition_elderly.xls`, how many people are vegetarians (no meat, no fish)?

- *File Intima_Media_Thickness.xls:*

- 5.1- Add a column BMI to the data.frame, with the BMI of each individual in the data file.
- 5.2- Retrieve the thickness of intima for the people with a $BMI > 30$.
- 5.3- Extract the “athletic” women.
- 5.4- Extract the “non-obese” people aged 50 or over (obese=BMI>30).

- *File bmichild.xls:*

- 5.1- Add a column BMI.
- 5.2- Extract the children with a $BMI < 15$ and an age ≤ 3.5 years.
- 5.3- How many such children are there?

- *File Birth_weight.xls:*

- 5.1- Add a variable PTL1 (number of children born before term), with three modalities (where the third modality, coded 2, corresponds to “2 or more” preterm births).
- 5.2- Same question with FVT (number of visits to a physician), to add FVT1.
- 5.3- Sort the file by increasing weight at birth (BWT).
- 5.4- Extract the individuals whose mothers are black or white and smoke.

B- Handling Missing Values

Import into a data.frame the following file:

<http://www.biostatisticien.eu/springeR/Infarction.xls>

- 5.1- Which rows include missing values?
- 5.2- Which individuals have more than one missing value?
- 5.3- Which variables include missing values?
- 5.4- Give at least one solution to remove all rows of this data.frame which include at least one missing value. In addition to logical operators and the extracting function, you are only allowed to use:

- (a) The functions `is.na()`, `prod()`, `apply()` and `as.logical()`
- (b) The functions `is.na()`, `apply()` and `any()`
- (c) The functions `is.na()`, `apply()` and `all()`
- (d) The function `complete.cases()`
- (e) The function `na.omit()`

C- Handling Character Strings

- 5.1- Import the file www.biostatisticien.eu/springeR/dept-pop.csv into a data.frame called dept.
- 5.2- Replace the first column with two new columns: one called numdep with the French *département* numbers and another with the names.

D- Influenza Epidemics in France Since 1984¹

- 5.1- Import the file <http://www.biostatisticien.eu/springeR/flu.csv> into a data.frame called flu. Make sure that you are handling missing values correctly.
- 5.2- Type `names(flu)`. As you can see, `flu$Date` includes dates in the format year (with century; for example, 2003) followed by the week number (two digits).
- 5.3- Determine the list of possible week numbers (hint: use the functions `sort()`, `substring()` and `unique()`).
- 5.4- First, you need to retrieve these dates in R in an object of class `POSIXlt`, for example, with the function `strptime()`. Using Table 5.3, and this function, transform the first (oldest) date into the POSIX format.
- 5.5- The data are in fact updated every Monday, since the first week. Determine which is the oldest date (Day, Month, Year) for which observations exist (hint: use the calendar <http://sentiweb.fr/calendrier.php>).
- 5.6- You should notice that there is a difference with the answer to question 5.4. To solve this problem, try adding "1" at the end of the first date and transforming it again with the function `strptime()`.
- 5.7- Display the ten first dates from the data files. Use the hint from the previous question to transform them with the function `strptime()`. Is the last date correct? If not, do you have an idea to solve this problem?
- 5.8- At this point, you should realize that the format of the dates in this file is not compatible with the POSIX format (which takes week numbers between 00 and 53). It is therefore not possible to directly use the function `strptime()` or `as.POSIXlt()` to transform these dates into an object type easy to handle by R. Type in the instruction
`date1 <- as.POSIXlt("Day,Month,Year", format="to be specified")`
where you will replace *Day*, *Month* and *Year* with the oldest date and *to be specified* with the relevant date format.
- 5.9- Type in the instruction `date1` then `date1+7`. What do you notice?
- 5.10- Find a way to add seven days to `date1` (hint: how many seconds are there in a day?).
- 5.11- Now, create the vector `dates` containing all the dates in the POSIX format, sorted from most ancient to newest (hint: use the function `nrow()`).

¹ Source: <http://www.sentiweb.fr/?lang=en>

- 5.12-** Use the function `substring()` on the vector `dates` to replace the first column of the data.frame `flu` with dates in the format "year-month-day" (for example, "1992-12-07").
- 5.13-** Use the vector `dates` and what you have learnt about extraction to select only the portion of the data.frame `flu` for dates between September 15, 1992 and November 3, 1993. Store this sub-table in an object called `portion`.
- 5.14-** Calculate the number of cases of influenza over this period for each French region (hint: pay attention to missing values; use the argument `na.rm`). Store the results in a vector called `flucases`.

E- Combining Tables or Lists; Other Manipulations

- 5.1-** Input into R the two following tables (check the row names):

```
> a
  [,1] [,2]
1     1    4
2     2    5
6     3    6
> b
  [,1] [,2]
3     1    5
4     2    6
5     3    7
7     4    8
```

- 5.2-** Combine `a` and `b` into a new table called `ab` containing

```
> ab
  [,1] [,2]
1     1    4
2     2    5
3     1    5
4     2    6
5     3    7
6     3    6
7     4    8
```

(hint: use `rbind()` and `order()`).

- 5.3-** Concatenate the elements of `list1` as columns of one matrix:

```
> list1 <- list()
> list1[[1]] <- matrix(runif(25),nr=5)
> list1[[2]] <- matrix(runif(30),nr=5)
> list1[[3]] <- matrix(runif(15),nr=5)
```

(hint: use the function `unlist()` or the function `do.call()`).

- 5.4-** Concatenate the elements of `list2` as rows of one matrix:

```
> list2 <- list()
> list2[[1]] <- matrix(runif(25),nc=5)
> list2[[2]] <- matrix(runif(35),nc=5)
> list2[[3]] <- matrix(runif(15),nc=5)
```

(hint: use the function `unlist()` or the function `do.call()`).

- 5.5-** Automatically select the diseases which have `tobacco` as a risk factor:

```
> tmp
      Disease      RiskFactors
1 Infarction tobacco, alcohol
2 Hepatitis         alcohol
3 Lung cancer        tobacco
```

(hint: use the function `grep()`).

Creating Functions

F- The French Chevalier de Méré

The French chevalier de Méré was a keen gambler. He was particularly fond of two gambles. In the first one, he would throw a die four times and bet that a 6 would come out at least once. In the second one, he would throw two dice 24 times and bet that a double-6 would come out. He had noticed that the first gamble is “beneficial”: there is more than a 50 % chance that a 6 will come out at least once. He thought that the second gamble is also beneficial.

- Propose code for a function called `fourthrows()` which returns 1 if there is at least one 6 out of 4 throws of a die and 0 otherwise. Do not use any loops.
- Propose code for a function called `twentyfourthrows()` which returns 1 if there is at least one double-6 out of 24 throws of two dice and 0 otherwise. Do not use loops.
- Propose code for a function called `meresix()` to confirm the chevalier de Méré’s intuition. It should use the first two functions and take a formal parameter `nsim` which gives the number of repetitions of the gamble.

Hint: use function `sample()`.

Chapter 6

R and Its Documentation

Prerequisites and goals of this chapter

- Chap. 3.
- This chapter presents the various ways to get help on the R software.

— SECTION 6.1 —

Integrated Help

6.1.1 The Command `help()`

R includes an online help. It is very complete and very well structured for all functions and for the various symbols in the language. There are several ways to access the help files; the main method is `help()`. It is used in command line mode.

For example, type:

```
help(help)
```

There is an alias for the command `help()`: the question mark `?`.

```
?sum  
?sd  
?"+"  
?"["[["
```

Warning

Sometimes, this alias will not work. In those cases, you will need to use the function `help()` with quotation marks.



```
?function      # Does not work.
help(function) # Returns an error.
help("function") # Correct call.
```

Let us look at the help of the function `mean()`.

```
?mean
① mean                  package:base          R Documentation
② Arithmetic Mean
③ Description:
   Generic function for the (trimmed) arithmetic mean.
④ Usage:
   mean(x, ...)
   ## Default S3 method:
   mean(x, trim = 0, na.rm = FALSE, ...)
⑤ Arguments:
   x: An R object. Currently there are methods for
       numeric dataframes, numeric vectors and dates.
       A complex vector is allowed for 'trim = 0', only.
   trim: the fraction (0 to 0.5) of observations to be trimmed
         from each end of 'x' before the mean is computed.
   na.rm: a logical value indicating whether 'NA' values
         should be stripped before the computation proceeds.
   ...: further arguments passed to or from other methods.
⑥ Value:
   For a data frame, a named vector with the appropriate
   method being applied column by column.
   If 'trim' is zero (the default), the arithmetic mean of
   the values in 'x' is computed.
   If 'trim' is non-zero, a symmetrically trimmed mean is
   computed with a fraction of 'trim' observations deleted
   from each end before the mean is computed.
⑦ References:
   Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988)
   _The New S Language_. Wadsworth & Brooks/Cole.
⑧ See Also:
   'weighted.mean', 'mean.POSIXct'
⑨ Examples:
   x <- c(0:10, 50)
   xm <- mean(x)
   c(xm, mean(x, trim = 0.10))
   mean(USArrests, trim = 0.2)
```

Here are the sections of this help file:

- ① The header of the file, with:
 - The name of the function: `mean`
 - The name of the package in which the function is included: `base`
 - The origin of the help file: `R Documentation`
- ② An explicit title for the function: `Arithmetic Mean`
- ③ A brief description of what the function does: `Description`
- ④ How to use the function; in particular, the compulsory and optional arguments:
`Usage`
- ⑤ A description of the function's arguments: `Arguments`
- ⑥ Explanations on the output of the function: `Value`
- ⑦ References (statistical articles or books) related to the function's application domain: `References`
- ⑧ The `See Also` section, which lists similar or related functions
- ⑨ Examples of use: `Examples`

Warning

Most help files follow this format. Make sure you understand and remember the structure of help files. You should also take the habit of checking the online help whenever you meet an unknown function, so as to understand its arguments and use.



Tip

Note that help files do not include graphs, for example, those that could be produced with the code in the section `Examples` ⑨. This would be interesting, especially for all the graphical functions. One way to get them is the function `example()`. You can also browse the website *R Graphical Manual*: <http://bm2.genes.nig.ac.jp/RGM2/index.php> which includes all R help files in HTML. In those files, when there are graphs, they are directly included in the section `Examples`.



6.1.2 Some Complementary Commands

In addition to the main command, `help()`, a few other complementary functions can be useful when looking for help on a given command. They are listed here:

- `help.start()`: this function opens a web browser with links to handbooks in HTML, help on functions included in all R packages (also HTML), a FAQ

(Frequently Asked Questions), and a search engine of the help files. There are also other more technical documents.

Linux



Under Linux, once you have entered the command `help.start()`, using the command `help()` will always result in the help being displayed in the web browser, rather than in the command line. To cancel this behaviour, use the instruction `options(htmlhelp = FALSE)`. To change browser (e.g., `firefox`), use the instruction `options(browser="firefox")`.

- `help.search()` or `??()`: this function is useful when you do not know the name of a command. It returns a list of functions (and the package in which they are included) related to your request. Try: `help.search("mean")`.
- `apropos()`: this instruction returns the names of functions which are a (potentially partial) match to the calling argument. For instance, `apropos("mean")` returns the names of functions containing the word `mean`.

Advanced users



Note also that the function `methods()` returns all the methods (functions) associated with an object. For instance, try `methods(summary)`.

- `library(help=package)`: this command lists all functions included in a package. It gives the same results as the command `help(package="package")`. We advise you to try the following instructions to list the main functions in R:

```
library(help=base)
library(help=utils)
library(help=datasets)
library(help=stats)
library(help=graphics)
library(help=grDevices)
```

Tip



The function `library(lib.loc = .Library)` returns the list of all packages (or libraries) installed on the system.

Conversely, the instruction `find("function")` indicates in which package a function is included.

```
> find("t.test")
[1] "package:stats"
```

- **vignette()**: vignettes are small PDF files which explain some notions in further detail. Type `vignette()` to get a list of vignettes, and for example `vignette("xtableGallery")` to open the PDF vignette of the package `xtable`.

Mac

All vignettes can also be read in a special vignette browser, from the menu “Help/Vignettes”. In this browser, you can open PDF files as well as R source code (as .R files) and consult directly the code of examples included in the vignette.



These three other functions might also be useful:

- **data()**: this command lists all datasets included in R.
- **example()**: this instruction lists examples of use of a function. For example, `example(mean)` executes the instructions included in the section *Examples* of the help file `help(mean)`.
- **demo()**: this instruction is similar to `example()`, but is only available for a small number of functions. When it is available, it shows the range of possible uses of a function. For example, try `demo(graphics)`.

— SECTION 6.2 —

† Help on the Web

The official R website (<http://www.r-project.org>) includes a huge amount of information about this software. You should spend some time exploring it. The following sections list other sources of information.

6.2.1 Search Engines

There are two main search engines for R:

- <http://search.r-project.org/nmz.html>

Tip

The command `RSiteSearch()` can be used to send a request on this website directly from R. The information is then displayed in your browser.

- <http://www.rseek.org>

There is also a very interesting collaboratively edited question and answer site for programmers available at URL <http://stackoverflow.com/questions/tagged/r>.

6.2.2 Message Boards

There are many message boards about R, where you can ask your questions. One message board with a lot of traffic is <http://r.789695.n4.nabble.com>.

6.2.3 Mailing Lists

A mailing list is a specific kind of e-mail, which sends messages to a large number of subscribers.

There are several mailing lists about R. The main ones are:

- <https://stat.ethz.ch/mailman/listinfo/r-help>
- <http://blog.gmane.org/gmane.comp.lang.r.general>
- <http://www.r-project.org/mail.html>
- R-announce: <https://stat.ethz.ch/mailman/listinfo/r-announce>

The website <http://r-project.markmail.org> can be used to search the archives of these lists.

You need to follow a few rules to post a message on these lists, as described here: <http://www.r-project.org/posting-guide.html>.

Mac

A list dedicated to Mac users: <https://stat.ethz.ch/mailman/listinfo/r-sig-mac>.

6.2.4 Internet Relay Chat (IRC)

IRC (*Internet relay chat*) is a real-time messaging service. You can use it to chat with other Internet users on predefined themes. The IRC channel on the R software is called (#R) on the freenode server.

To access it, you can either use client-side software such as xchat (www.xchat.org) or use your browser through websites such as <https://webchat.freenode.net>.

To connect to this channel using xchat, type in these instructions:

```
/server irc.freenode.net  
/join #R
```

6.2.5 Wiki

A *wiki* is a website where pages can be freely edited by visitors. *Wikis* are used to aid collaborative writing with minimal constraints.

There is a *wiki* about R here: <http://rwiki.sciviews.org>.

SECTION 6.3

† Literature About R

6.3.1 Online

Literature about R is available online in many forms:

- **Task Views:** lists of packages useful in a given domain, grouped by themes. A website describing Task Views is available at the URL <http://cran.r-project.org/web/views>.
- **Frequently Asked Questions (FAQ):** Frequently Asked Questions about R are listed here: <http://cran.r-project.org/faqs.html>.
- **Specialized journals:** two online journals deal with the R software: the *R Journal*, previously known as *R News* (<http://journal.r-project.org>), and the *Journal of Statistical Software* (<http://www.jstatsoft.org>).

- **Handbooks:** many handbooks are available as a PDF on the R website: <http://cran.r-project.org/other-docs.html>

6.3.2 Printed Material

Many books have been published about R recently. We find the following to be the most interesting:

- **Data Analysis and Graphics Using R: An Example-Based Approach** [26]
- **The R Book** [12]
- **Statistics and Data with R** [10]
- **Software for Data Analysis: Programming with R** [8]
- **Lattice: Multivariate Data Visualization with R** [36]
- **R for SAS and SPSS Users** [32]
- **Introductory Statistics with R: An Applied Approach Through Examples** [13]
- **A First Course in Statistical Programming with R** [6]
- **A Handbook of Statistical Analyses Using R** [15]
- **A Beginner's Guide to R** [42]
- **R Cookbook** [39]
- **R in a Nutshell** [1]
- **The Art of R Programming** [28]
- **The R Inferno** [7]

Memorandum

`help()`, `?()`: get help on a function or a symbol
`help.search()`: list of functions relevant to your request
`apropos()`: list of function names which include the request
`library(help=package)`: list of all functions in a package
`data()`: list of all datasets available in R
`example()`: execute the *Examples* section of the corresponding help file
`demo()`: launch a small demonstration of the possible uses of a function
`vignette()`: open a PDF file with details on a function
`help.start()`: open the HTML version of the R help files
`RSiteSearch()`: start a request on the official R website search engine



Exercises

- 6.1-** Which R instruction should you type to get help on the function `mean()`?
- 6.2-** Explain the purpose of the command `apropos()`.
- 6.3-** Explain the purpose of the command `example()`.
- 6.4-** Explain the purpose of the command `RSiteSearch()`.
- 6.5-** How is a help file structured?
- 6.6-** Which command would you use to get the list of functions available in the package `stats`?
- 6.7-** Explain how to display a dataset available in R.



Worksheet

Where to Find Information

- 6.1-** Find the R function which lists all combinations of k elements out of n .
- 6.2-** Use this function to list all combinations of three elements out of `c(5, 8, 2, 9)`.
- 6.3-** Find the dataset available in R which gives the rates of violent crimes in the USA.
- 6.4-** Describe the contents of this dataset.
- 6.5-** Subscribe to the mailing list <https://stat.ethz.ch/mailman/listinfo/r-help>.
- 6.6-** Read the rules to follow before asking a question (<http://www.r-project.org/posting-guide.html>).
- 6.7-** Find out how to unsubscribe from the mailing list.
- 6.8-** Using the method of your choice, join the IRC channel R and start a polite conversation with channel members.

- 6.9-** Register on the message board <http://r.789695.n4.able.com>.
- 6.10-** Read the R FAQ for Microsoft Windows. Try to understand the meaning of *TAB completion*.
- 6.11-** Use TAB completion to list all files in the current directory.

Chapter 7

Drawing Curves and Plots

Prerequisites and goals of this chapter

- Reading previous chapters.
- This chapter describes the graphical possibilities of R, but does not go as far as expert graphical functions such as `hist()`, `barplot()` and so on. These functions are described in Chap. 11. We show generic ways to make adjustments to most plots you can draw.

— SECTION 7.1 —

Graphics Windows

7.1.1 Basic Graphics Windows, Manipulation and Saving

All plots created in R are displayed in special windows, separate from the console. They are called “R graphics: Device *device-number*”, where *device-number* is an integer giving the number of the window (or device).

To open a graphics window, use the command `windows()` or `win.graph()`.

This command takes several arguments. Some are briefly described in the following table:

<code>width</code>	Width of the graphics window, in inches
<code>height</code>	Height of the graphics window, in inches
<code>pointsize</code>	Default font size
<code>xpinch, ypinch</code>	Double. Pixels per inch, horizontally and vertically
<code>xpos, ypos</code>	Integer. Position of the upper left corner of the window, in pixels

Linux

Under Linux, the command is `X11()` instead of `windows()`.

Mac

On a Macintosh, the command is `quartz()`.

When several graphics windows are open, only one is “active”. This is the window in which all graphical events occur. Each window is associated with a device number; the console is number 1.

Here are a few functions to manipulate graphics windows, using their device number.

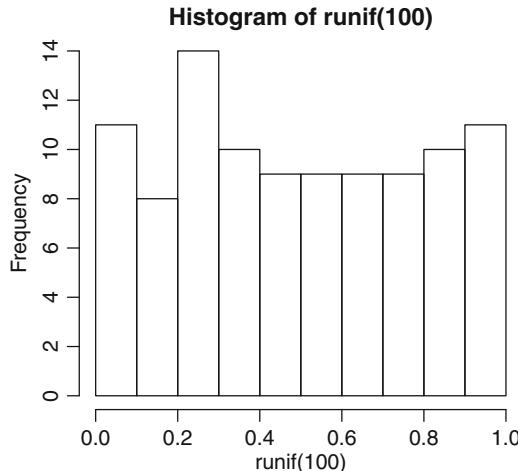
<code>dev.off(device-number)</code>	Close window <i>device-number</i> (if no device number is specified, the current active window is closed).
<code>graphics.off()</code>	Close all open graphics windows.
<code>dev.list()</code>	Return the device numbers of open graphics windows.
<code>dev.set(device-number)</code>	Activate window <i>device-number</i> .
<code>dev.cur()</code>	Return the device number of the active window (1 for the console).

Note that once a plot has been drawn, it can be saved to a file with the command `savePlot()` as follows:

```
savePlot(filename="Rplot",
         type=c("wmf", "png", "jpeg", "jpg", "bmp",
                "ps", "pdf"), device=dev.cur())
```

The argument `filename` is the name of the file under which the plot should be saved; `type` is the file type (Windows metafile, PNG, JPEG, BMP, PostScript or PDF) and `device` is the device number of the window with the plot to be saved (by default, the active window). Note that the available file types may depend on your operating system.

```
> hist(runif(100))
> savePlot(filename="mygraph.png", type="png")
```



Two other instructions can be used in this context:

- `dev.copy2eps(file="mygraph.eps")`
- `dev.copy2pdf(file="mygraph.pdf")`

which respectively create Postscript and PDF files.

Tip

There are other functions to save your plots under useful formats: `postscript()`, `pdf()`, `pictex()`, `xfig()`, `bitmap()`, `bmp()`, `jpeg()`, `png()`, `tiff()`. However, they are used in a slightly different way: first, type the name of one of these instructions, then draw your plot, and finish by calling `dev.off()`. Note that using these commands does not display the plot on the screen.



```
> pdf(file="mygraph.pdf")
> hist(runif(100))
> g <- dev.off()
```

7.1.2 Splitting the Graphics Window: `layout()`

If you want to draw several plots in the same window, R offers the possibility of splitting that window in as many boxes as needed.

A first possibility is the function `par()` with the argument `mffrow` (read the warning in Sect. 7.7 about the function `par()`). For instance, the following example splits the graphics window into three rows and two columns. Every time you call the

drawing of a function, one of the small boxes is filled, row by row (the command `mfcoll` is used to fill column by column), as shown in Fig. 7.1 below.

```
> par(mfrow=c(3,2))
```

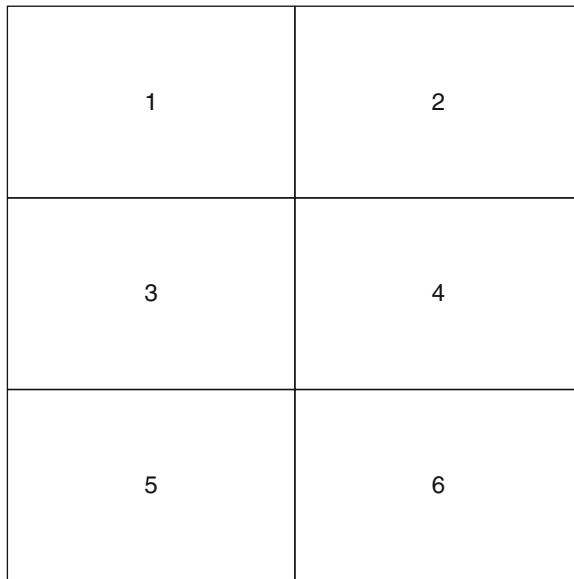


Fig. 7.1: Effect of argument `mfrow` of function `par()`. Numbers have been added to gain a better understanding of where future plots will be drawn

The function `layout()` is used to get a more sophisticated split than with the function `par()`. The following example shows how this splitting is specified in an intuitive way, thanks to the argument `mat`, to draw five separate plots (Fig. 7.2).

```
> mat <- matrix(c(2,3,0,1,0,0,0,0,4,0,0,0,0,5),4,3,byrow=TRUE)
> mat
 [,1] [,2] [,3]
[1,]    2    3    0
[2,]    1    0    0
[3,]    0    4    0
[4,]    0    0    5
```

```
> layout(mat)
```

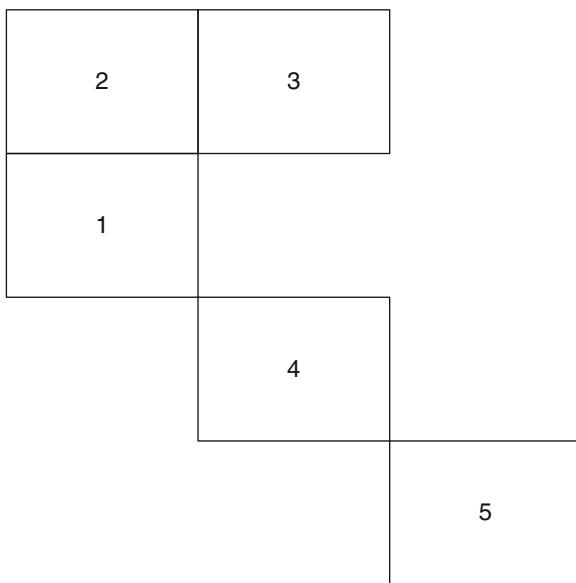


Fig. 7.2: Potential of the function `layout()`

Tip

To display the previous figure in R, use the instruction `layout.show(5)`.



Every time you call the drawing of a function, the plots are displayed in order in the numbered boxes, in increasing order of the box numbers (Fig. 7.3).

Also, note that with the argument `widths`, you can specify the respective relative widths of the columns of `mat`. The same can be done for the heights of rows with the argument `heights`.

```
> layout(mat, widths=c(1,5,14), heights=c(1,2,4,1))
> layout.show(5)
```

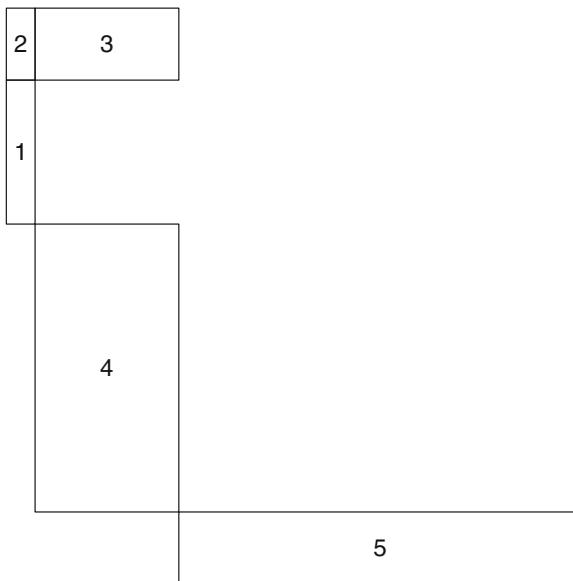


Fig. 7.3: The function `layout()` and its arguments `widths` and `heights`

SECTION 7.2

Low-Level Drawing Functions

7.2.1 The Functions `plot()` and `points()`

The function `plot()` is the generic function to draw plots. It takes as input argument the coordinates of the points to draw (Fig. 7.4).

Warning

You can also use the function `plot()` on an R object for which a graphical method is defined. Examples are given in Chaps. 14 and 15.

Here are the most useful arguments of this function.

Argument	Description
x	Vector of x coordinates of points to draw.
y	Vector of y coordinates of points to draw.
type	Specify the type of plotting: "p" for points, "l" for lines, "b" for both, "c" for empty points joined by lines, "o" for overplotted points and lines, "h" for vertical lines, "s" for stair steps and "n" to plot nothing (but to display the window, with axes).
main	Specify the main title.
sub	Specify the subtitle.
xlab	Specify the label of the x axis.
ylab	Specify the label of the y axis.
xlim	Vector of length 2. Specify the lower and upper bound for the x axis.
ylim	Vector of length 2. Specify the lower and upper bound for the y axis.
log	Character string which contains "x" (respectively "y", "xy" or "yx") if the x axis (respectively the y axis, both) is to be logarithmic.

```
> plot(1:4,c(2,3,4,1),type="b",main="Main title",
+   sub="Subtitle",xlab="Label for x",ylab="Label for y")
```

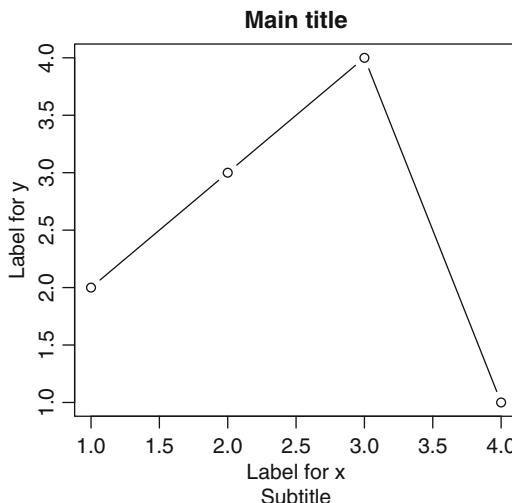


Fig. 7.4: The function `plot()`

Note that successive calls of the function `plot()` create a new plot every time, which replaces the previous one (unless the graphics window has been split, as explained above). The function `points()` can remediate this issue by overlaying the new plot on top of the old one. It takes the same arguments as `plot()` (Fig. 7.5).

```
> points(1:4,c(4,2,1,3),type="l")
```

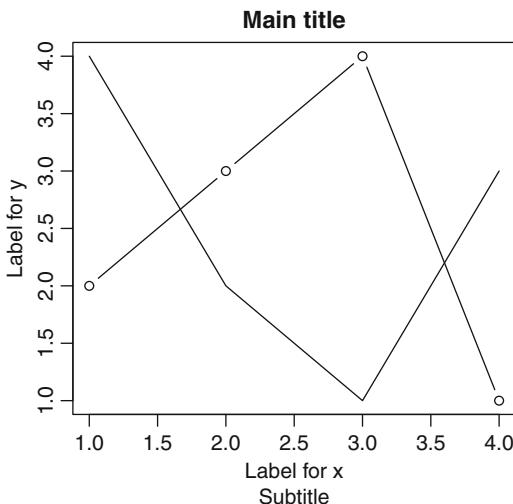


Fig. 7.5: The function `points()`

Tip

The function `approx()` provides a linear or constantwise interpolation between points.

7.2.2 The Functions `segments()`, `lines()` and `abline()`

The functions `segments()` and `lines()` are used to join points with line segments, added on to a pre-existing plot (Fig. 7.6).

```
> plot(0,0,"n")
> segments(x0=0,y0=0,x1=1,y1=1)
> lines(x=c(1,0),y=c(0,1))
```

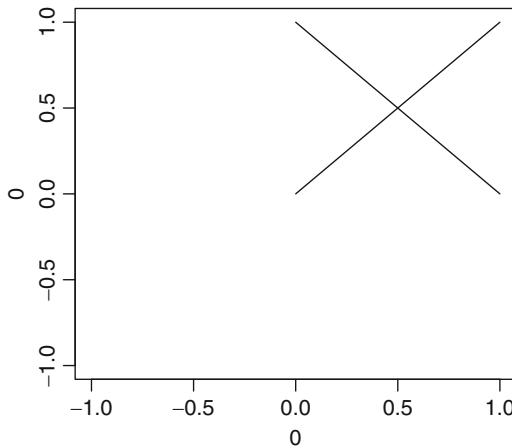


Fig. 7.6: The functions `segments()` and `lines()`

The function `abline()` is used to draw a straight line of equation $y = a + bx$ (specified by the arguments `a` and `b`) or a horizontal (argument `h`) or vertical (argument `v`) line (Fig. 7.7).

```
> plot(0,0,"n"); abline(h=0,v=0); abline(a=1,b=1)
```

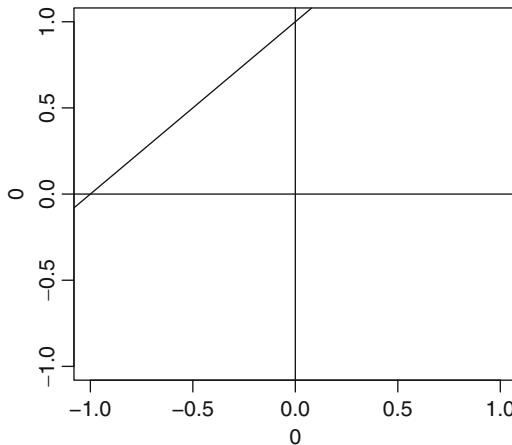
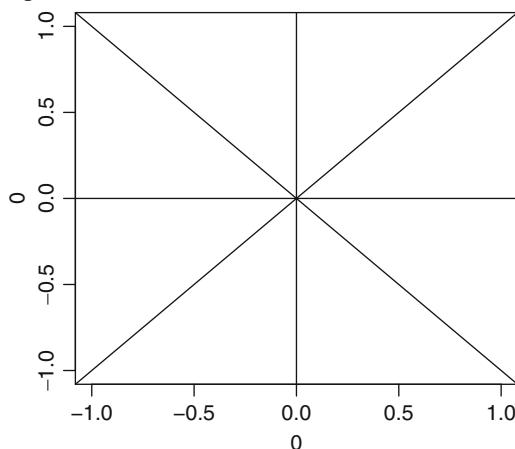


Fig. 7.7: The function `abline()`

Do it yourself

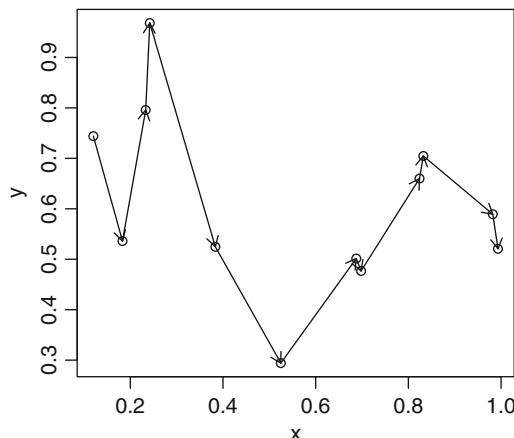
Reproduce this plot.



7.2.3 The Function arrows()

This function is used to draw arrows between pairs of points. It takes an argument `length` to indicate the size of the arrowhead (Fig. 7.8):

```
> x <- runif(12); y <- runif(12)
> i <- order(x,y); x <- x[i]; y <- y[i]
> plot(x,y)
> s <- seq(length(x)-1)
> arrows(x[s], y[s], x[s+1], y[s+1], length=0.1)
```

Fig. 7.8: The function `arrows()`

7.2.4 The Function `polygon()`

As the name suggests, this function is used to draw polygons and to fill in a polygon with a specified colour.

Do it yourself



Enter the following command in the R console:

```
example(polygon)
```

Tip

The command `polygon(locator(10, "l"))` is used to draw a ten-edge polygon by clicking in the graphics window on the points where the polygon vertices should be.



7.2.5 The Function `curve()`

This function is used to draw a curve in a Cartesian coordinate system, on the interval specified by the bounds `from` and `to`.

```
> curve(x^3 - 3*x, from=-2, to=2)
```

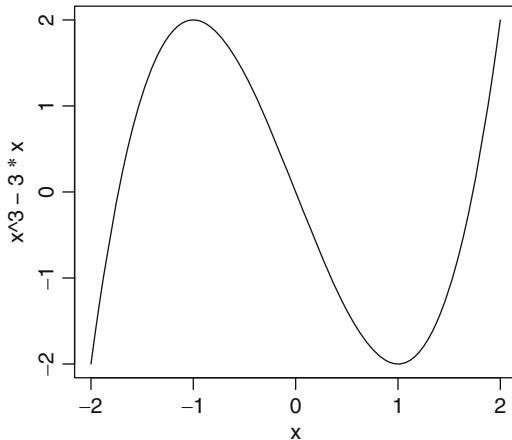


Fig. 7.9: The function `curve()`

Note that the argument `add=TRUE` can be used to indicate that the curve should be overlaid on a pre-existing plot (Fig. 7.9).

Do it yourself



Use the following instruction to draw the density histogram of 10,000 random values from a normal distribution:

```
hist(rnorm(10000), prob=TRUE, breaks=100)
```

Use the function `curve()` to overlay on top of this histogram the density function of a $\mathcal{N}(0, 1)$ distribution, given by the function `dnorm()`.

7.2.6 The Function `box()`

This function is used to add a box around the current plot. The argument `bty` manages the type of box; the argument `lty` manages the type of line used to draw the box. Note that by default, the function `plot()` adds a box, unless it is given the argument `axes=FALSE` (Fig. 7.10).

```
> plot(runif(7), type = "h", axes = FALSE)
> box(lty = "1373")
```

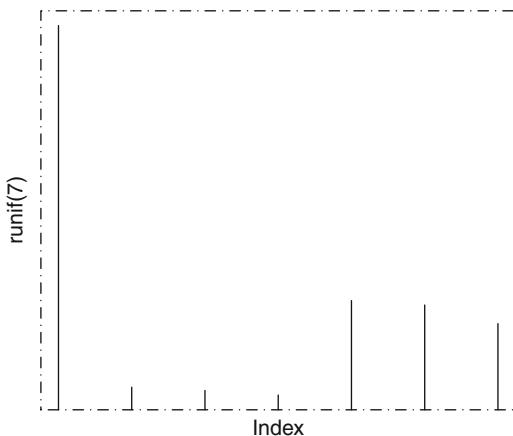


Fig. 7.10: The function `box()`

SECTION 7.3

Managing Colours

7.3.1 *The Function `colors()`*

This function returns the names of the 657 colours known to R.

Tip

If you want to get the different shades of orange, you can use the instruction

```
> colors()[grep("orange", colors())]
[1] "darkorange"   "darkorange1"  "darkorange2"  "darkorange3"
[5] "darkorange4"  "orange"      "orange1"     "orange2"
[9] "orange3"      "orange4"     "orangered"   "orangered1"
[13] "orangered2"  "orangered3"  "orangered4"
```



These colours can be used in your plots, for example, with the argument `col` of the function `plot()` (Fig. 7.11).

```
> plot(1:10,runif(10),type="l",col="orangered")
```

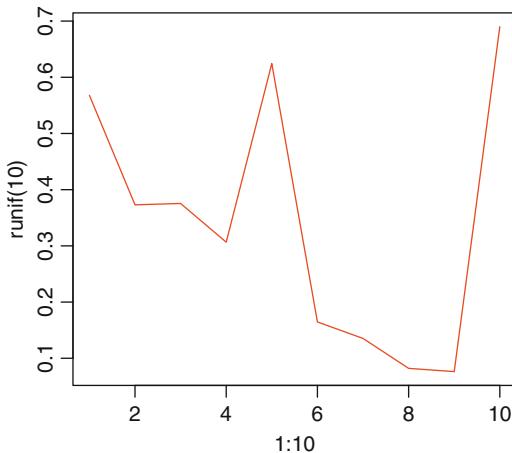


Fig. 7.11: The argument `col` of function `plot()`

See also

Note that you can also change the colour of the other elements of the plot, such as the axes or the title. To this end, refer to Sect. 7.7, page 176, which deals with the function `par()`.

7.3.2 Hexadecimal Colour Coding

R gives the possibility of using hexadecimal colour coding, for example, with the argument `col` of the function `plot()`. Each colour is coded as its decomposition into three base colours: red, green and blue. Each component can take a value between 0 and 255 (0: complete absence of the colour; 255: saturation of the colour). Hexadecimal coding of these 256 values gives codes between 00 and FF.

Here are few examples of colours:

```
Black:  #000000
White: #FFFFFF
Almond green: #82C46B
Lemon yellow: #F7FF3C
Peacock blue: #048B9A
Midnight blue: #10076B
```

Note that you can use the function `rgb()` to get the hexadecimal code of a colour from its decomposition into red, green and blue.

```
> rgb(red=26,green=204,blue=76,maxColorValue = 255)
[1] "#1ACC4C"
> rgb(red=0.1,green=0.8,blue=0.3)
[1] "#1ACC4D"
```

The function `col2rgb()` does the reverse operation:

```
> col2rgb("#1ACC4C")
[,1]
red     26
green   204
blue    76
```

You can even get transparency, with the argument `alpha` of the function `rgb()`, as seen in Fig. 7.12:

```
> curve(sin(x),lwd=30,col=rgb(0.8,0.5,0.2),xlim=c(-10,10))
> curve(cos(x),lwd=30,col=rgb(0.1,0.8,0.3,alpha=0.2),add=TRUE)
```

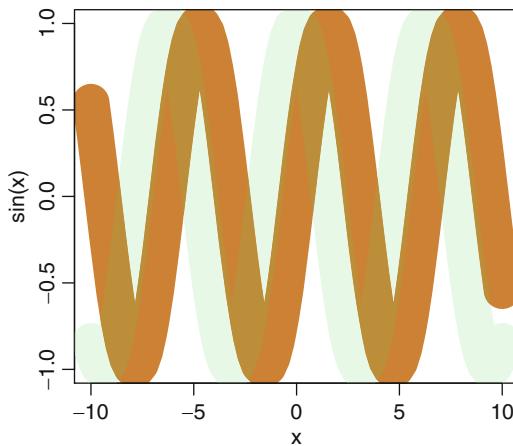


Fig. 7.12: The argument `alpha` of function `rgb()`

If your graphics card allows it, R can thus handle up to 256^3 colours or over 16 million colours. The next example uses the function `rainbow()` and should give you an idea of this range (Fig. 7.13).

```
> pie(rep(1, 200), labels = "", col = rainbow(200), border = NA)
```

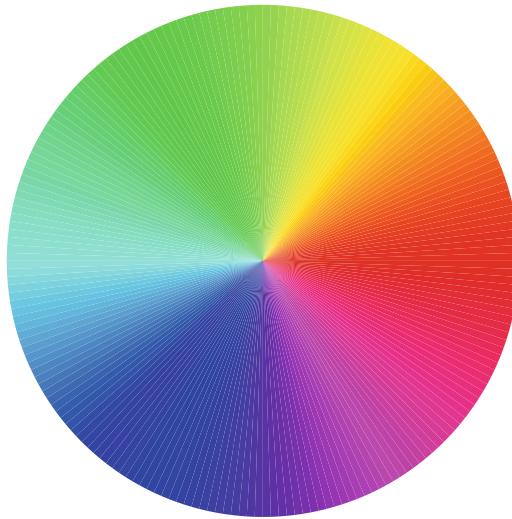


Fig. 7.13: An example using function `rainbow()`

You can also add to **R** the package `RColorBrewer`. This package can be used to automatically create ideal colour palettes for beautiful presentations: shades of a colour, complementary or diverging colours (Fig. 7.14).

7.3.3 *The Function `image()`*

This function creates and displays a grid of coloured or greyscale rectangles. The rectangles are also called pixels (*picture elements*). It can be used to display 3D or spatial data, *i.e.* images (Fig. 7.15).

```
> X <- matrix(1:12,nrow=3)
> X
 [,1] [,2] [,3] [,4]
[1,]    1     4     7    10
[2,]    2     5     8    11
[3,]    3     6     9    12
```

The numbers in the boxes were added using the function `text()` which is introduced later on (Fig. 7.16).

Warning

Beware of how the coloured rectangles are organized in Fig. 7.15: left to right and bottom to top. It is therefore an anticlockwise 90 degrees rotation of the display of the contents of the matrix X.



```
> require("RColorBrewer")
> display.brewer.all()
```

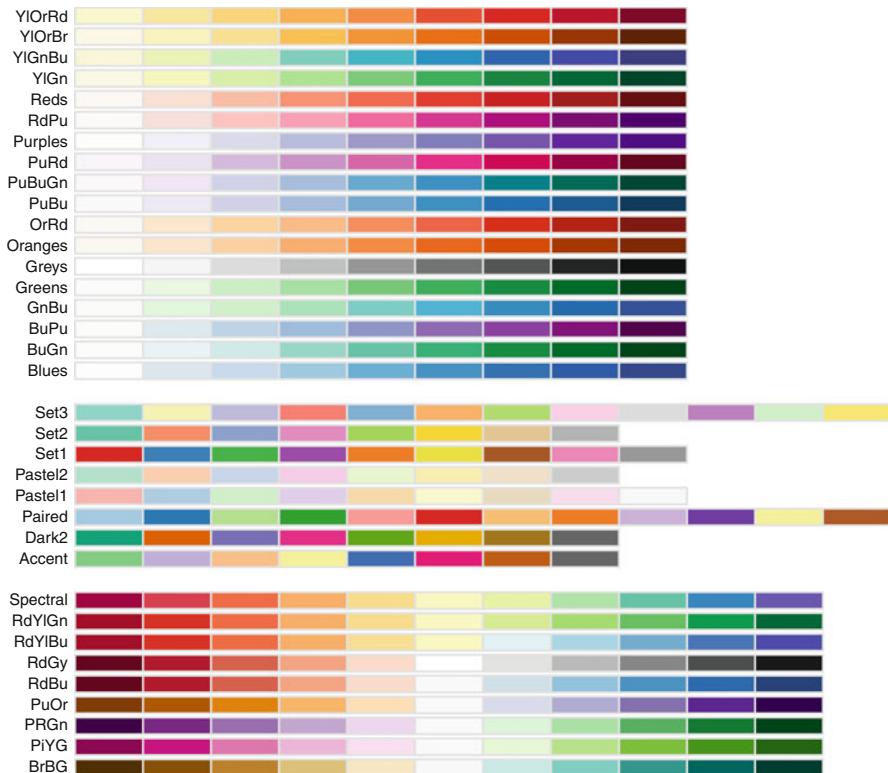
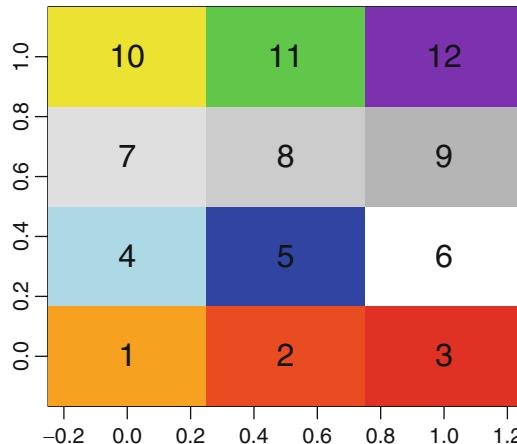


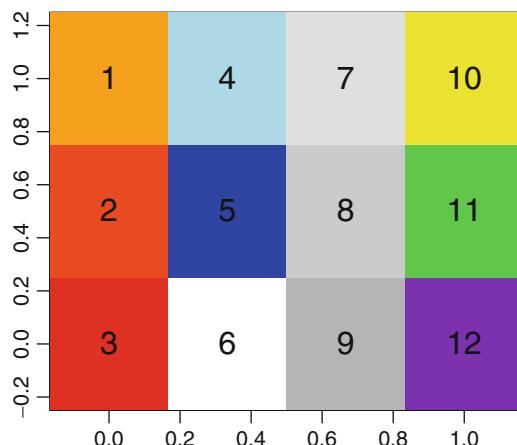
Fig. 7.14: The function `display.brewer.all()` from package `RColorBrewer`

You can get a display coherent with how the data are organized in the matrix X as follows:

```
> colours <- c("orange","orangered","red","lightblue",
+           "blue", "white","lightgrey","grey",
+           "darkgrey","yellow","green","purple")
> image(X,col=colours)
> text(rep(c(0,0.5,1),4),rep(c(0,0.3,0.7,1),each=3),1:12,cex=2)
```

Fig. 7.15: The function `image()`

```
> image(as.matrix(rev(as.data.frame(t(X)))),col=colours)
> text(rep(c(0,0.33,0.67,1),each=3),rep(c(1,0.5,0),4),1:12,cex=2)
```

Fig. 7.16: The function `image()` with a coherent display of the data

Do it yourself

Install and load the package `caTools`. Use the function `read.gif()` of this package to read the file <http://www.biostatisticien.eu/springerR.R.gif>. Use the function `image()` to display it in R. Use the colours given by `read.gif()` and display the image in the correct orientation.

SECTION 7.4

Adding Text**7.4.1 The Function `text()`**

This function is used to add text to a plot. A very interesting feature is that it can also be used to add mathematical formulae. In addition to the string itself, you need to specify the x and y coordinates of the centre of the string. To display a mathematical expression, use the function `expression()`. The `bquote()` function can also be useful (Fig. 7.17).

```
> plot(1:10,1:10,xlab=bquote(x[i]),ylab=bquote(y[i]))
> text(3,6,"some text")
> text(4,9,expression(widehat(beta) == (X^T * X)^{-1} * X^T * y))
> p <- 4; text(8,4,bquote(beta[.(p)])) # Combining "math" and
                                         # numerical variables.
```

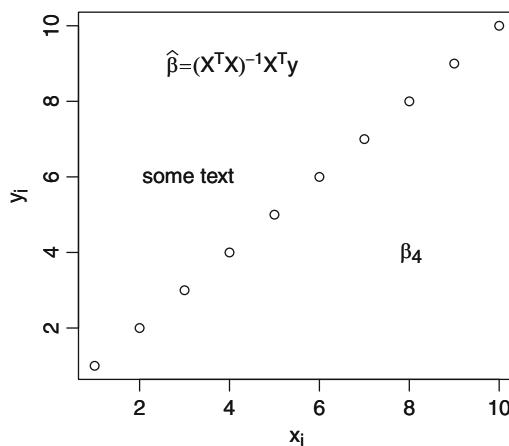


Fig. 7.17: The function `text()`

Tip

Use the command `demo(plotmath)` to see the various possibilities of adding mathematical expressions to a plot. This will also show the relevant commands.

Do it yourself

Plot a point at the coordinates (1, 1). Then use the function `text()` to add the text "ABC", also at the coordinates (1, 1). Observe the effect of the argument `pos`, which takes the values 1 (below), 2 (left), 3 (above) or 4 (right).

7.4.2 The Function `mtext()`

This function is used to add text in the margins of the graphics window. It can also be used to add mathematical formulae.

It takes an argument `side` (1=bottom, 2=left, 3=top or 4=right) to specify which margin the text should be added in (Fig. 7.18).

```
> plot(1:10,1:10)
> mtext("bottom",side=1)
> mtext("left",side=2)
> mtext("top",side=3)
> mtext(expression(x^2+3*y+hat(beta)),side=4)
```

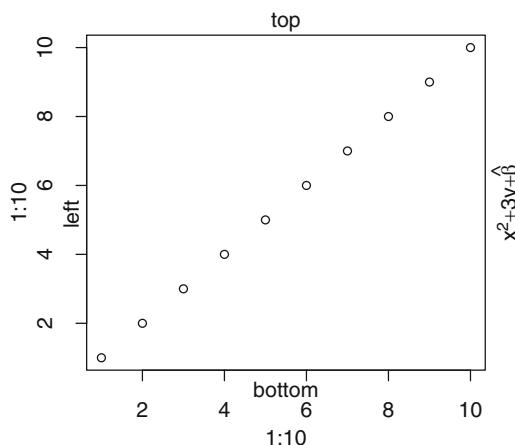


Fig. 7.18: The function `mtext()`

SECTION 7.5

Titles, Axes and Captions

7.5.1 The Function `title()`

This function is used to add titles to a plot: a main title above the plot with the argument `main`, a subtitle below the plot with the argument `sub`, a label for the `x` axis with the argument `xlab` and a label for the `y` axis with the argument `ylab`. Note that these arguments can also be specified directly when calling graphical functions such as `plot()` (Fig. 7.19).

```
> plot.new()
> box()
> title(main = "Main title", sub = "Subtitle",
+           xlab = "x label", ylab = "y label")
```

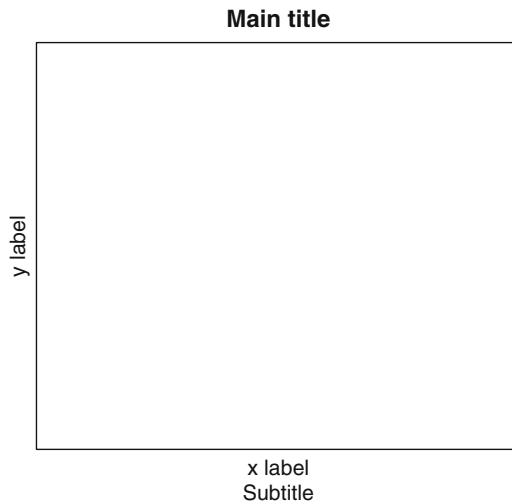


Fig. 7.19: The function `title()`

Tip

A title can be written on several lines, thanks to the carriage return character "\n" (Fig. 7.20).

```
> plot(1:10,main="Title on\n three\n lines",xlab="",ylab="")
```

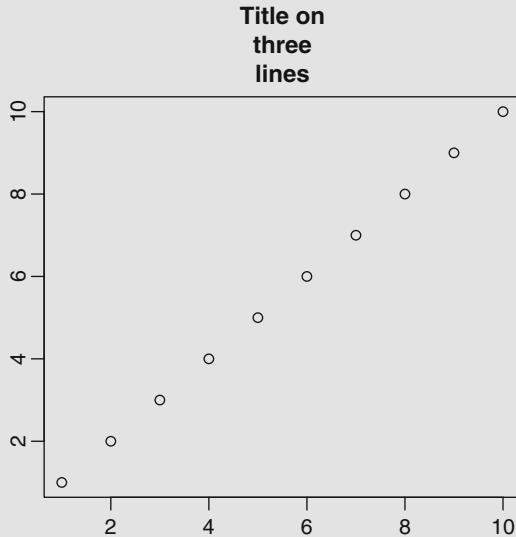


Fig. 7.20: Plot title on several lines

7.5.2 The Function `axis()`

This function adds an axis to a pre-existing plot. You can specify on which side to draw the axis, the position of ticks and several other arguments.

Note

In general, you will only need the function `axis` when you wish to control the details of the axes. To this end, you can first draw a plot (e.g., with the function `plot()`) without the axes, with the argument `axes=FALSE`.

Here are a few of the main arguments of the function `axis()` (Fig. 7.21).

Argument	Description
<code>side</code>	Specify on which side to draw the axis: <code>side=1</code> (below), <code>side=2</code> (left), <code>side=3</code> (above), <code>side=4</code> (right).
<code>at</code>	Specify where to draw the ticks.
<code>labels</code>	Either a Boolean to specify whether the ticks should be annotated or a character string specifying the annotation at the ticks.
<code>tick</code>	Boolean specifying whether the ticks should be drawn.
<code>col</code>	Colour of the axis.

Other arguments are available and are described in the online help.

```
> plot.new()
> lines(x=c(0,1),y=c(0,1),col="red")
> axis(side=1,at=c(0,0.5,1),labels=c("a","b","c"),col="blue")
```

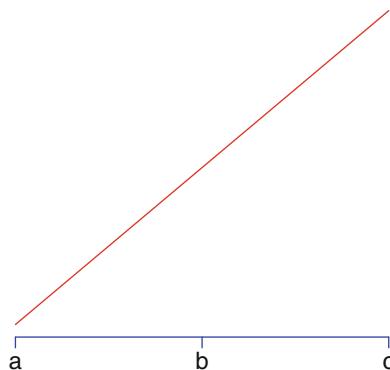


Fig. 7.21: The function `axis()`

7.5.3 The Function `legend()`

This function is used to add a caption to a pre-existing plot (Figs. 7.22, 7.23). Here are a few of its arguments:

Argument	Description
<code>x, y</code>	Specify the coordinates of the position of the caption on the plot.
<code>legend</code>	Vector of character strings or expressions to display in the caption.
<code>fill</code>	Vector of colours to fill the background of the caption box.
<code>lty, lwd</code>	Integer. Line type or line width for lines in the legend. You must specify one of these arguments to get lines in the legend.
<code>col</code>	Vector of colours of points or lines in the legend.

```
> plot(1:4,1:4,col=1:4)
> legend(x=3,y=2.5,legend=c("a","b","c","d"),fill=1:4)
```

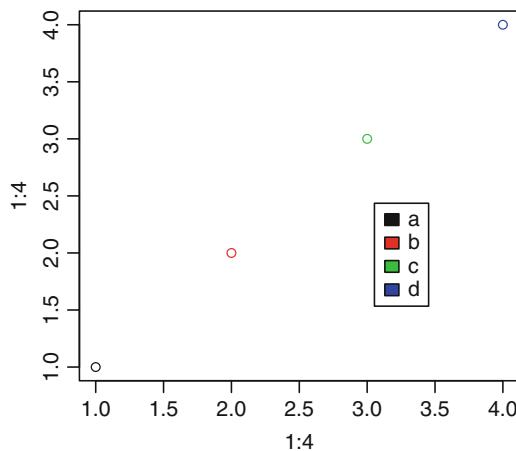


Fig. 7.22: The function `legend()` with squares

```
> plot(1:4,1:4,col=1:4,type="b")
> legend(x=3,y=2.5,legend=c("a","b","c","d"),col=1:4,lty=1)
```

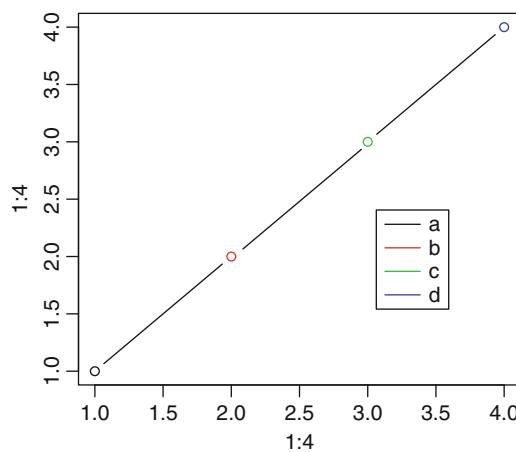


Fig. 7.23: The function `legend()` with line segments

Many other arguments are available and are described in the online help.

SECTION 7.6

Interacting with the Plot

7.6.1 The Function `locator()`

It is used to place a point on a plot or to get its coordinates with a click of the mouse. It can also be useful to add text (or a caption) at a specific location, thanks to the mouse.

Do it yourself



Enter the following instructions, then click anywhere on the plot you get:

```
plot(1,1)
text(locator(1),labels="Here") # Click on the graphics window.
```

7.6.2 The Function `identify()`

It is used to identify and mark points already present on a plot. The following practical should help you understand this function.

Do it yourself



Enter the following instructions, then click next to points on the plot. Use a right click to exit the interactive mode.

```
> plot(swiss[,1:2])
> x <- identify(swiss[,1:2],labels=rownames(swiss))
> x
```

SECTION 7.7

† Fine-Tuning Graphical Parameters: `par()`

The function `par()` takes many arguments to fine-tune your plots. Use this function to set (or query) general graphical parameters.

Here is how to use this instruction:

- `par(arg-name)` outputs the default value of the parameter *arg-name* of the function `par()`.
- `par(arg-name=val)` changes the value of the parameter *arg-name* to the value *val*.
- `par()` returns the list of all graphical parameters currently in use, as well as the current values.

Warning

Before changing the values of parameters of the function `par()`, you should save the old values. That way, you can restore them later if needed.



```
# Save the default values of par().
save.par <- par(no.readonly = TRUE)
# Now we can change some parameters.
par(bg="red")
# Then restore the old values.
par(save.par)
```

Before we present the detailed use of this function, it is worth noting that the graphics window (also called *device region*) includes the *figure region*, which in turn includes the *plot region*. Figure 7.24 illustrates this.

Here is an (almost complete) list of the various parameters of the function `par()`, along with a short description. We have organized them in groups to make it easier to find relevant parameters (Table 7.1).

• Managing the graphics window

Table 7.1: Parameters to **manage the graphics window**

Name	Description
<code>ask</code>	Boolean. If TRUE, the user is prompted to press ENTER before a new plot is drawn. Use <code>devAskNewPage()</code> instead.
<code>din*</code>	Dimensions <code>c(width, height)</code> of the graphics window, in inches (stands for <i>device region inches</i>).
<code>fig</code>	A numeric vector of the form <code>c(x1, x2, y1, y2)</code> giving the <i>normalized device coordinates</i> of the figure region, in which the plot will be drawn.
<code>fin</code>	A numeric vector of the form <code>c(width, height)</code> giving the size of the figure region, in inches (stands for <i>figure region inches</i>).
<code>mai</code>	A numeric vector of the form <code>c(bottom, left, top, right)</code> giving the size of the margins, in inches.
<code>mar</code>	A numeric vector of the form <code>c(bottom, left, top, right)</code> giving the number of margin lines on the four sides of the plot. The default value is <code>c(5, 4, 4, 2) + 0.1</code> .
<code>mex</code>	<code>mex</code> is an expansion factor for the size of the font used to describe the coordinates in the plot margins. Note that this does not change the font size, but rather specifies, as a multiple of <code>csi</code> , the font size to convert between <code>mar</code> and <code>mai</code> and between <code>oma</code> and <code>omi</code> . Its value is 1 when the device is opened and is reinitialized when the layout is changed (<code>cex</code> is also reinitialized).
<code>mfcol</code> , <code>mfrow</code>	A vector of the form <code>c(nl, nc)</code> . The successive plots (or <i>multi-figures</i>) will be drawn in a matrix of size <code>nl</code> -by- <code>nc</code> in the graphics window, filled respectively by -columns- (<code>mfcol</code>) or by -rows- (<code>mfrow</code>). Consider the alternatives: <code>layout()</code> and <code>split.screen()</code> .
<code>mfg</code>	A numeric vector of the form <code>c(i, j)</code> where <code>i</code> and <code>j</code> indicate the cell of the figures matrix in which the next plot should be drawn. The figures matrix must have been predefined with one of the parameters <code>mfcol</code> or <code>mfrow</code> .
<code>mpg</code>	The margin line (in units of <code>mex</code>) for the axis titles, labels and lines. The default value is <code>c(3, 1, 0)</code> .
<code>new</code>	Boolean, by default FALSE. If set to TRUE, the next high-level graphical command (in fact <code>plot.new()</code>) will not erase the old plot and will overlay the new plot.
<code>oma</code>	A vector of the form <code>c(bottom, left, top, right)</code> giving the size of the external margins (units: lines of text).
<code>omd</code>	A vector of the form <code>c(x1, x2, y1, y2)</code> giving the region inside the external margins, in NDC (= <i>normalized device coordinates</i>), i.e. as a proportion (in [0,1]) of the graphics window.
<code>omi</code>	A vector of the form <code>c(bottom, left, top, right)</code> giving the size of the outer margins, in inches.
<code>pin</code>	The dimensions of the plot region, in inches <code>c(width, height)</code> .
<code>plt</code>	A vector of the form <code>c(x1, x2, y1, y2)</code> giving the coordinates of the plot region as fractions of the current figure region.
<code>pty</code>	A character specifying the plot region type: "s" generates a square plot region and "m" generates the maximal plot region.
<code>usr</code>	A vector of the form <code>c(x1, x2, y1, y2)</code> giving the extreme values of the user coordinates of the plot region. If a log scale is used (i.e. <code>par("xlog")</code> is TRUE), then the limits in <code>x</code> are $10^{\text{par}("usr")}[1:2]$. The same goes for the limits in <code>y</code> .
<code>xpd</code>	A Boolean or NA. If FALSE, all plots are attached to the plot region. If TRUE, all plots are attached to the figure region. If NA, all plots are attached to the plot window. See also <code>clip()</code> .

* An asterisk has been added to parameters which cannot be modified by the user (read only)

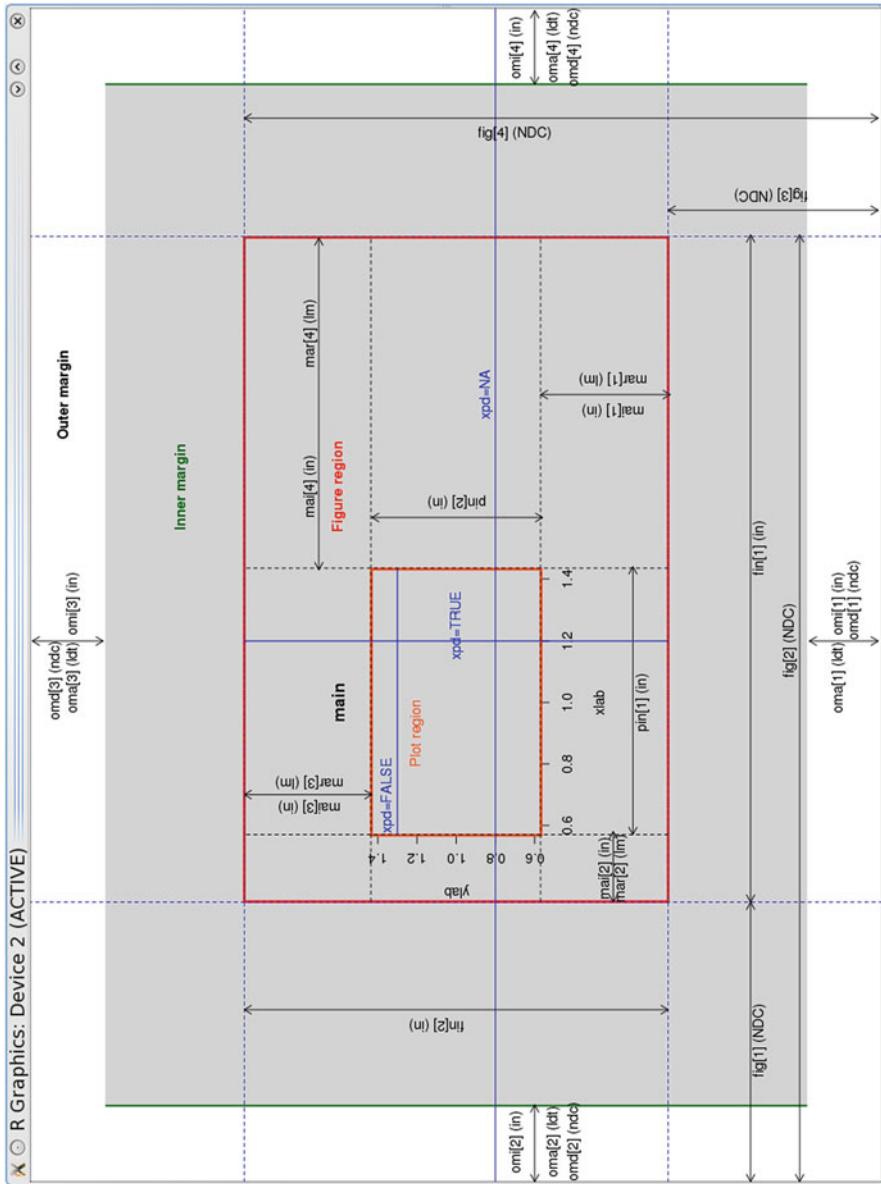


Fig. 7.24: Figure illustrating the fine management of graphical parameters

The previous plot should help you better understand some of these parameters (Fig. 7.24).

- **Managing colours**

Table 7.2: Parameters to **manage colours**

Name	Description
<code>bg</code>	Background colour of device region.
<code>col</code>	Colour of plot.
<code>col.axis</code>	Colour of axis annotations.
<code>col.lab</code>	Colour of x and y labels.
<code>col.main</code>	Colour of main title.
<code>col.sub</code>	Colour of subtitles.
<code>fg</code>	Colour of foreground (axes and box around the plot). Sets <code>col</code> to the same value.

See how the parameters in Table 7.2 can be put in use (Fig. 7.25):

```
> par(bg="lightgray",col.axis="darkgreen",col.lab="darkred",
+      col.main="purple",col.sub="black",fg="blue")
> curve(cos(x),xlab="xlab in darkred",main="Title in purple",
+        xlim=c(-10,10),sub="sub in black")
> curve(sin(x),col="blue",add=T)
```

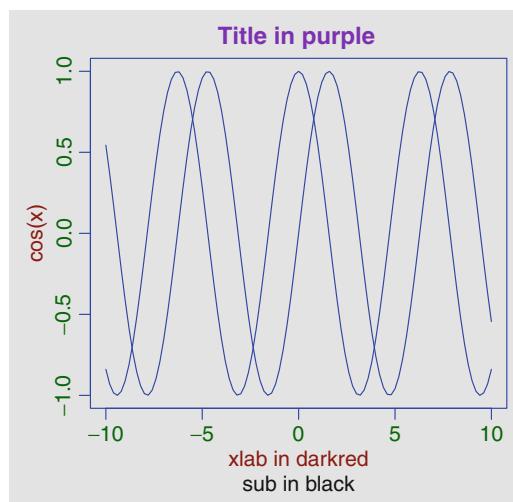


Fig. 7.25: Managing the colours of a plot

- **Managing text**

Table 7.3: **Managing text** displayed on a plot

Name	Description
adj	The value of adj determines how character strings are adjusted in <code>text()</code> , <code>mtext()</code> and <code>title()</code> . A value of 0 leads to left-adjusted text, 0.5 to centred text and 1 to right-adjusted text. Any value in [0, 1] is allowed, and some values outside this interval sometimes also work. Note that the argument adj to the function <code>text()</code> also allows <code>adj = c(x, y)</code> for different adjustments in x and in y . Note that for <code>text()</code> , the text is adjusted relatively to a point, whereas for <code>mtext()</code> and <code>title()</code> , it is adjusted relatively to the plot region or graphical window.
ann	If set to FALSE, high-level graphical functions will not add annotations to plots they produce (axes and main title). By default, annotations are added.
cex	A numeric value giving the character expansion coefficient for text and symbols on the plot, relatively to a reference value.
cex.axis	Character expansion coefficient for axes annotations.
cex.lab	Expansion for x and y labels.
cex.main	Expansion for main title.
cex.sub	Expansion for subtitles.
cin*	Size of characters in inches <code>c(width, height)</code> .
cra*	Size of characters in pixels <code>c(width, height)</code> .
crt	Numeric value specifying (in degrees) how various characters must be rotated. Must be a multiple of 90. Compare with <code>srt</code> which rotates character strings.
csi*	Height of characters in inches. Identical to <code>par("cin")[2]</code> .
cxy*	Size of the characters <code>c(width, height)</code> in units expressed relatively to the user coordinates. <code>par("cxy")</code> is equal to <code>par("cin")/par("pin")</code> times a scaling factor in the user coordinates. Note that <code>c(strwidth(ch), strheight(ch))</code> for a given character string <code>ch</code> is usually much more accurate.
family	Name of a font family. Maximal size is 200 bytes. Each plot device puts this name in relation with a description of the font specific to the device. The default value is "", which means that the default font is used (see the device help file for further details). Other oft-used values are <code>serif</code> , <code>sans</code> and <code>mono</code> ; Hershey fonts are also available. This can be specified in the function <code>text()</code> .
font	An integer specifying the font to use for text. In general, 1 corresponds to ordinary text, 2 to bold, 3 to italics and 4 to bold italics. 5 should be symbol font (Adobe encoding).
font.axis	Font for axis annotations.
font.lab	Font for labels of x and y axes.
font.main	Font for main title.
font.sub	Font for subtitles.
ps	Integer. Size of text, in points (but not of symbols).
srt	Character string rotation, in degrees. See the comment on <code>crt</code> . Only supported by <code>text()</code> .

* An asterisk indicates arguments which cannot be modified by the user (read only)

Here is an example of use of the arguments `adj` and `srt` (Fig. 7.26, Table 7.3):

```
> par(mfrow = c(1, 3))
> vals <- c(0, 0.5, 1)
> for (adj in vals) {
+ par(adj = adj)
+ plot(0, main = paste("adj =", adj), col.lab = "red",
+       col.main = "red", type = "n")
+ text(1, 0, "abc", col = "red", cex = 2)
+ abline(h = 0, lty = 2)
+ abline(v = 1, lty = 2)
+ }
> abline(v=0.8,h=-0.5,lty=2)
> text(0.8, -0.5, "abc", col = "red", cex = 2, adj=c(0,1))
> abline(h=0.5,v=0.5,lty=2)
```

```
> text(0.8, 0.5, "ABC", col="red", cex=2, adj=c(0.5,0.5), srt=120)
```

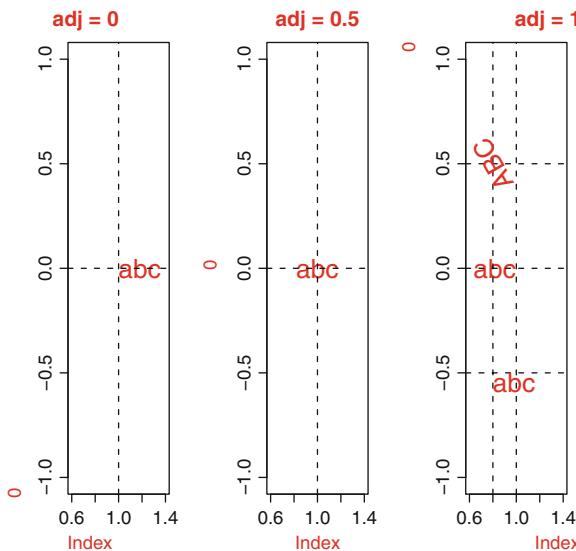


Fig. 7.26: Example of use of the arguments `adj` and `srt`

This second example shows how to use different fonts (Fig. 7.27, Table 7.3):

```
> par(cex.axis=1.5)
> plot(1:5,y=rep(1,5),type="n",font.axis=2,font.lab=3,xlab=
+ "xlab in italics",ylab="",font.main=4,main="Title in bold
+ italics", font.sub=5,sub="Subtitle in symbol font")
> text(2,1.2,"Normal text")
> par(ps=30)
> text(3,1,"A Hershey font",family="HersheyScript")
> par(ps=14)
```

```
+ family="HersheyGothicEnglish")
```

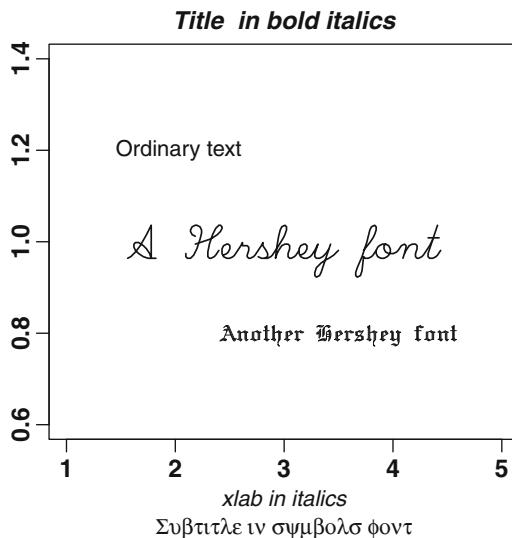


Fig. 7.27: Using different fonts on a plot

Tip

To see all symbols and fonts available in R, use the following command:

```
demo(Hershey)
```

- **Managing axes**

Table 7.4: Parameters to **manage axes**

Name	Description
<code>bty</code>	Character string to specify the type of box around the plot (the axes). If <code>bty</code> is one of "o" (default value), "l", "t", "c", "u" or "]", the box looks like the corresponding character. A value of "n" hides the box.
<code>lab</code>	Numeric vector of the form <code>c(x, y, 1en)</code> modifying how the axes are annotated. The values of <code>x</code> and <code>y</code> (approximately) specify the number of ticks on the <code>x</code> and <code>y</code> axes. <code>1en</code> specifies the size of the labels. The default value is <code>c(5, 5, 7)</code> . Note that this only affects the values of the arguments <code>xaxp</code> and <code>yaxp</code> when the coordinates system is put in place and is not used when the axes are drawn. <code>1en</code> is not implemented yet.
<code>las</code>	Number in the set {0,1,2,3}. Style of the axis labels. 0=always parallel to the axis (default), 1=always horizontal, 2=always orthogonal to the axis, 3=always vertical. Note that the argument <code>srt</code> of <code>par()</code> , which manages the rotation of character strings, does not affect axis labels.
<code>tck</code>	Length of tick marks on axes, as a fraction of the minimum of the height and width of the drawing region. If <code>tck >= 0.5</code> , then it is interpreted as a fraction of the relevant side, and if <code>tck=1</code> , then a grid is drawn. The default (<code>tck = NA</code>) sets <code>tcl = -0.5</code> .
<code>tcl</code>	Length of the tick marks as a fraction of the height of a line of text. The default value is <code>-0.5</code> . Entering <code>tcl = NA</code> sets <code>tck = -0.01</code> .
<code>xaxp</code>	A vector of the form <code>c(x1, x2, n)</code> giving the coordinates of the extreme ticks and the number of intervals between ticks, when <code>par("xlog")</code> is FALSE. Otherwise, when the scale is logarithmic, the three values have different meanings. See the online help for details. See also <code>axTicks()</code> .
<code>xaxs</code>	Style for the intervals on the <code>x</code> axis. Possible values: "r", "i", "e", "s", "d". The style is usually controlled by the data range or by <code>xlim</code> if it is specified. The style "r" (<i>regular</i>) first enlarges the data range by 4 % on both sides and then finds an axis with pretty labels which fits in the range. The style "i" (<i>internal</i>) only finds an axis with pretty labels which fits the original data range. The style "s" (<i>standard</i>) finds an axis with pretty labels in which the original data range is included. The style "e" (<i>extended</i>) is similar to the style "s", but it also leaves space to draw symbols inside the bounding box. The style "d" (<i>direct</i>) specifies that the current axes must be used for subsequent plots. As of writing, only the styles "r" and "i" are implemented.
<code>xaxt</code>	A character to specify the type of <code>x</code> axis. A value of "n" implies that an axis is created, but not drawn. The standard value is "s".
<code>xlog</code>	Boolean (see <code>log</code> in <code>plot.default()</code>). If TRUE, a logarithmic scale is used (e.g., after <code>plot(*, log = "x")</code>). For a new graphics window, the default is FALSE, i.e. a linear scale.
<code>yaxp</code>	A vector of the form <code>c(y1, y2, n)</code> giving the coordinates of the extreme tick marks and the number of intervals between these ticks, except for the logarithmic scale. See <code>xaxp</code> above.
<code>yaxs</code>	Style for the intervals on the <code>y</code> axis. See <code>xaxs</code> above.
<code>yaxt</code>	Character to specify the type of <code>y</code> axis. A value of "n" implies that the axis is created, but not drawn.
<code>ylog</code>	Boolean; see <code>xlog</code> above.

A few of these parameters are used in the following example (Fig. 7.28):

```
> # Enlarge the bottom margin to leave space for the x
# labels.
> par(mar = c(7, 4, 4, 2) + 0.1)
> # Define a box style, ten ticks in x and y,
# horizontal labels, and graduations of length 1 (which gives
# a grid)
> par(bty="7", col="blue", lab=c(10,10,1), las=1, tck=1)
> # Create a plot without the x axis and without
# x labels.
> plot(1 : 8, xaxt = "n", xlab = "")
> # Add the x axis with ticks only.
> axis(1, labels = FALSE)
> # Create the label vector.
> labels <- paste("Label", 1:8, sep = " ")
> # Add the x labels to the default ticks.
> text(1:8, par("usr")[3] - 0.25, srt = 45, adj = 1,
+       labels = labels, xpd = TRUE)
> # Add a subtitle at the bottom, on the sixth margin line (out
# of 7).
> mtext(1, text = "Labels for the X axis", line = 6)
```

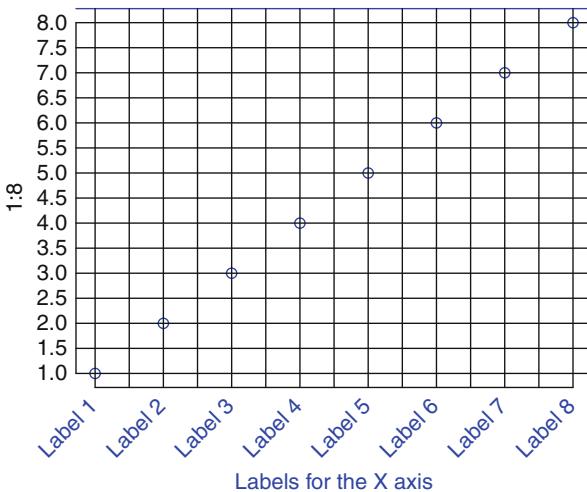


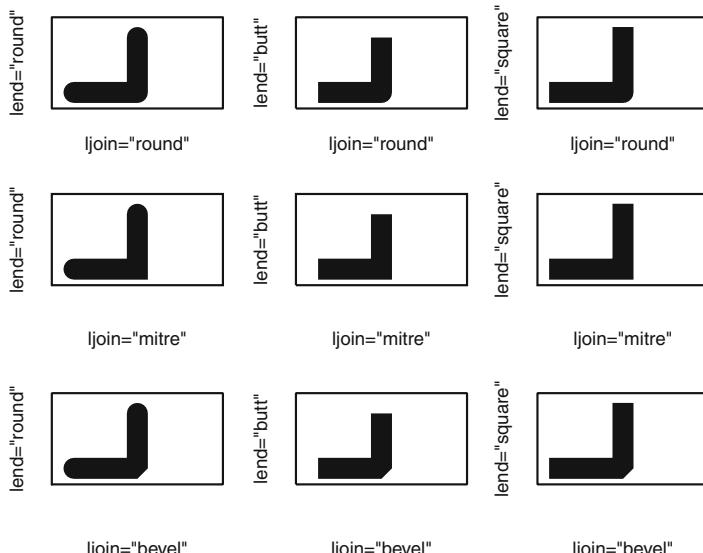
Fig. 7.28: Labels on a plot

• Lines and symbols

Table 7.5: Parameters for **lines and symbols**

Name	Description
<code>lend</code>	End-of-line style. Can be specified with an integer or a character string: 0 or "round" means that a semicircle is added at the end of a line; 1 or "butt" means that lines end straight; 2 or "square" means that a small square is added at the end of a line.
<code>lheight</code>	Line height multiplier. When a text extends over several lines, the height of the space between lines is found by multiplying the height of characters both by the current character expansion factor and by the line height multiplier. The default value is 1. Used in <code>text()</code> and <code>strheight()</code> .
<code>ljoin</code>	Line join style. Can be specified with an integer or a character string: 0 or "round" means a round join (default); 1 or "mitre" means a straight join; 2 or "bevel" means bevelled line joins.
<code>lmitre</code>	Controls when straight joins are automatically converted to pointy joins. Must be greater than 1 and the default value is 10. Does not work on some peripherals.
<code>lty</code>	Line type. Can be specified by an integer (0=blank, 1=solid, 2=dashes, 3=dots, 4=dots and dashes, 5=long dashes, 6=two dashes) or by one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash" or "twodash". Note that "blank" uses invisible lines (so does not draw them). You can also give a character string (of length no more than 8) giving the length of the solid and empty segments. See the section Line type specification of the online help.
<code>lwd</code>	Line width in the plot (positive number), defaults to 1.
<code>pch</code>	Either an integer specifying a symbol or a character to replace small circles in point plots.

The following plot should help you better understand the arguments `lend` and `ljoin` (Fig. 7.29, Table 7.5).

Fig. 7.29: The arguments `lend` and `ljoin`

The figure below shows the different symbols you can obtain with the argument pch. The type of points in a plot is controlled with the argument pch. Points 0 to 20 are of the same colour, controlled with the argument col. Points 21 to 25 also have a filling colour, controlled with the argument bg of the function points() (Fig. 7.30).

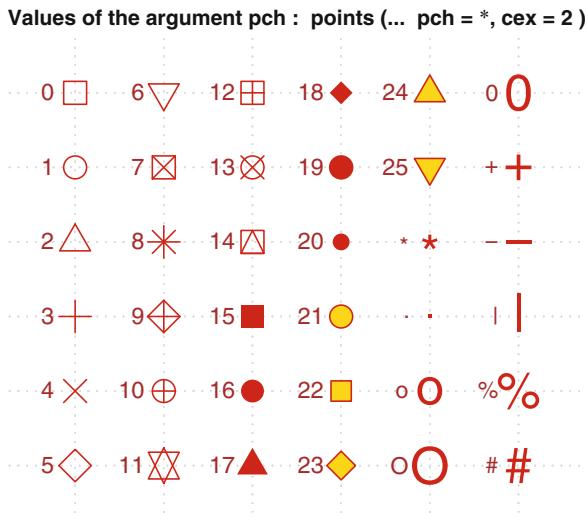


Fig. 7.30: The argument pch

Figure 7.31 shows how to use the arguments lty and lwd:

```
> plot(1,1,type="n")
> for (i in 0:6) abline(v=0.6+i*0.1,lty=i,lwd=i)
> abline(v=1.3,lty="92",lwd=10)
```

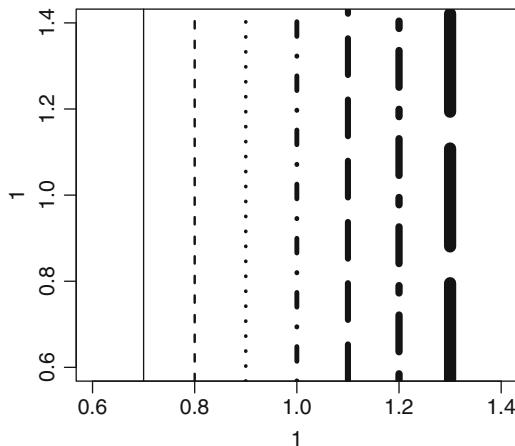


Fig. 7.31: The arguments lty and lwd

SECTION 7.8

† Advanced Plots: rgl, lattice and ggplot2

There are other R packages to manage plots in a more advanced way. Space does not allow us to describe these packages in much detail. We shall simply give a few striking examples, in the hope that the advanced reader will want to find out more.

- Package **rgl**

This package is used to create pretty 3D plots, with interactive viewpoint navigation using the mouse. Try the following commands to get an idea:

```
require("rgl")
demo(rgl)
example(rgl)
```

- Package **lattice**

An entire book is dedicated to this package: [36]. The following example shows that in the package **lattice**, plots can be considered as objects (as in object-oriented programming), which some readers might find pleasing. For example, suppose that you have used the following commands to draw a graph and that you realize you made a mistake in the title.

```
x <- 1:100
y <- sin(x)
plot(x,y,type="l",main="Cosine plot")
```

The way to resolve this issue would be to redraw the entire figure, this time with the correct title.

With the package **lattice**, you can avoid this hassle.

```
require("lattice")
xyplot(y~x,type="l",main="Cosine plot")
```

The following instruction can be used to change the title without redrawing the plot!

```
update(trellis.last.object(),main="Sine plot")
```

- Package **ggplot2**

We shall only mention the package **ggplot2**, which makes explicit the conceptual links between plots and statistical analyses. You can visit the website of this package at <http://ggplot2.org> and the website of the book devoted to it at <http://ggplot2.org/book>.

Memorandum

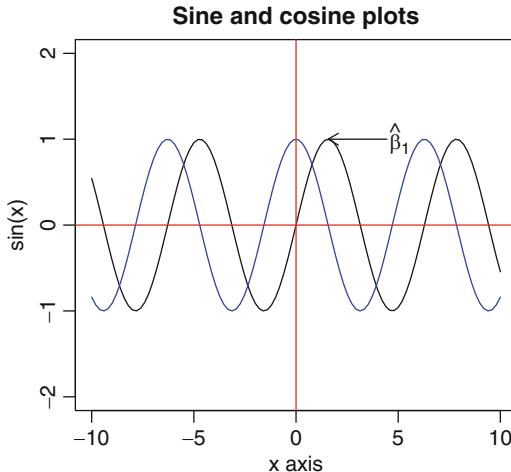
`dev.off()`: close the active graphics window
`savePlot()`: save the contents of the active graphics window to a file
`layout()`: split the graphics window into boxes
`plot()`: draw points and optionally lines between them
`points()`: add points to a pre-existing plot and optionally lines between them
`segments(), lines(), abline()`: add lines to a plot
`arrows()`: add an arrow to a plot
`polygon()`: draw a polygon
`curve()`: draw a curve, specified by its equation
`box()`: add a box around the active plot
`colors()`: return the list of colour names known to R
`text()`: add text or mathematical symbols to a plot
`mtext()`: add text to the margins of a plot
`title()`: manage the titles of a plot
`axis()`: add an axis to a plot
`legend()`: add a caption to a plot
`locator()`: detect the coordinates of a point on a plot with a click of the mouse
`identify()`: identify a pre-existing point on a plot
`par()`: advanced management of all graphical parameters



Exercises

- 7.1-** What is the command `windows()` used for? And the command `dev.off()`?
- 7.2-** Suppose you drew a plot with the command
`curve(cos(x))`. Which R instruction would you use to save this plot as a PDF in a file called `myplot.pdf`?
- 7.3-** Give a detailed explanation of the effect of the instruction
`par(mfrow=c(3,2))`.
- 7.4-** What is the function `layout()` used for?
- 7.5-** Which command would you use to add a scatter plot to a pre-existing plot?
- 7.6-** Which argument of the function `plot()` would you use to get points with dashes between them?
- 7.7-** Name a function which draws a straight line.
- 7.8-** What is the function `curve()` used for?
- 7.9-** Which argument would you use to manage the colours in a plot?
- 7.10-** Which function would you use to display an image? Give an instruction to display an image, the values of which are given in a matrix X, so that the output is coherent with the way X is displayed in the console.
- 7.11-** Which function would you use to add text to a plot?
- 7.12-** Which function would you use to find the coordinates of a point in a plot with a click of the mouse?
- 7.13-** Give a detailed explanation of the effect of the instruction `par(ask=TRUE)`.

- 7.14- Which argument of the function `par()` would you use to specify the type of line drawn by the function `curve()`?
- 7.15- Which argument of the function `par()` would you use to display other symbols instead of small circles in a scatter plot?
- 7.16- Give a list of instructions to display the following plot. The central axis system must be displayed in red. The cosine curve must be displayed in blue.



Worksheet

Creating Various Plots

A- Complex Numbers

- 7.1- Reproduce the plot on complex numbers in Chap. 3 (page 47).

B- Flag of Canada

- 7.1- Install the package `caTools`.
- 7.2- Use the function `read.gif()` to read the image <http://www.biostatisticien.eu/springeR/canada.gif>.
- 7.3- Display the image with the function `image()`.
- 7.4- Redraw this flag in another window using the functions `plot()`, `rect()` and `polygon()` (hint: use the function `locator()`).

C- Frequency Tables

The following table represents scores of burning sensation for 16 subjects in a study to test a new hydrogel bandage. The first column gives the subject number. The next columns give the score of burning sensation (on a scale from 1 to 4) for weeks 1 (W1) to 7 (W7).

Nr	W1	W2	W3	W4	W5	W6	W7
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	2
3	1	1	1	1	1	2	3
4	1	1	1	1	1	3	4
5	1	1	1	1	2	3	3
6	1	1	1	1	1	1	1
7	1	1	1	3	4	2	2
8	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1
10	1	1	1	1	1	1	4
11	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1
13	1	2	1	3	2	3	4
14	1	1	1	2	2	4	4
15	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1

We shall propose an interesting way to display these data.

- 7.1- For week W7, calculate the vector $(f_1, 1-f_1, f_2, 1-f_2, f_3, 1-f_3, f_4, 1-f_4)$ where f_i is the frequency of modality i ($1 \leq i \leq 4$) observed in week W7 over the 16 subjects. (Hint: use the functions `tabulate()`, `cbind()`, `t()` and `as.vector()`).
- 7.2- Now, use the function `apply()` to do the same calculation for all other weeks. Store the result in a matrix.
- 7.3- Use the function `barplot()` and the argument `col=c("black", "white")` on this matrix. The plot you get gives an overview of the evolution of the variable Burning sensation over time.
- 7.4- Change the previous plot so that the bars representing frequencies are in red. Week numbers should be in blue and at the top of the plot instead of the bottom. Modality numbers should be on the left, in blue. Add a title to the plot.

D- Anatomic Images of the Brain

Data acquired during magnetic resonance imaging (MRI) of the brain are usually stored as a binary file with extension `*.img`. We shall see how to read and display such data.

- 7.1- Import the file [http://www.biostatisticien.eu/springeR/anat!\[\]\(92d6065c34fa5c6c5d3b9045cb889f8a_img.jpg\)](http://www.biostatisticien.eu/springeR/anat.img), which contains the image of a single brain section of 256×256 pixels,

using the function `readBin()`. These data can be treated as a sequence of 256×256 byte pairs (`raw`). Store these data in an object called `bytes`.

- 7.2- When the data were recorded, each pair of bytes was in fact written in reverse (e.g., the pair `02 56` was recorded as `56 02`). You therefore need to permute each byte pair. Store the result of this operation in `x`.
- 7.3- This sequence of byte pairs now needs to be transformed into numeric values which can be displayed graphically. With two bytes (such as `02 56`), you can use the instruction `as.numeric("0x0256")` to get the corresponding decimal value (in this case, the result is 598). Transform `x` into decimal values and store the result in an object called `values` (hint: use the functions `matrix()`, `apply()` and `paste()`).
- 7.4- Recreate the matrix of size 256×256 containing the observations stored in `values`. Call this matrix `X`.
- 7.5- Use the function `image()` on `X`. Use a colour gradient of shades of grey with about one hundred shades, using the function `gray()`.
- 7.6- Note that the R package `AnalyzeFMRI` does exactly that. After downloading the two files <http://www.biostatisticien.eu/springeR/anat.img> and <http://www.biostatisticien.eu/springeR/anat.hdr>, and after installing package `AnalyzeFMRI`, you could have got the exact same result by typing

```
require("AnalyzeFMRI")
Y <- f.read.volume("/path/to/anat.img") # Replace path.
image(X,col=gray(0:1000 / 1000))
```

E- Drawing the Map of a Region of France

The package `maps` includes maps of various countries. We shall use it to draw the borders of a French *département*.

- 7.1- Install and load the packages `maps` and `mapdata`.
- 7.2- Draw the map of France: `map("france")`.
- 7.3- Get the data on borders of French regions:
`france <- map("france",plot=FALSE)`
- 7.4- Display the contents of the object `france` and make sure you understand how it is organized. For example, note that latitude and longitude data are stored in `france$x/france$y` for each *département* in `france$names` (until the next NA).
- 7.5- Create a vector `indNA` containing the indices of missing values.
- 7.6- Create an object containing the name of the *département* of your choice (e.g., `depname <- "Gard"`).
- 7.7- Create an object called `inddept` containing the *département* index `depname` in the vector `france$names`.
- 7.8- Draw the map of your *département*.

- 7.9-** Add a point for a place on the map. You can get the coordinates (latitude and longitude) of a place on the website <http://www.gpsvisualizer.com/geocode>.

F- Representation of the Geoid in France

A geoid can be seen as a gravitation equipotential surface, going through the average sea level datum.

- 7.1-** Import the file <http://www.biostatisticien.eu/springeR/raf98.gra> in a matrix, using the function `scan()`. First, read the associated file <http://www.biostatisticien.eu/springeR/geoidformat.txt> which gives a description of the file format.
- 7.2-** Try to reproduce the plot available at <http://www.biostatisticien.eu/springeR/geoid.png>. Do not try to superpose the map of France yet (hint: use the functions `scan()`, `layout()`, `par()`, `image()`, `axis()`, `contour()`, `legend()` and `rainbow()`).

Chapter 8

Programming in R

Prerequisites and goals of this chapter

- Read all previous chapters first. A neophyte user can skim through this chapter on first reading. Indeed, it is well known that programming in a language requires a more advanced level than using a language.
- The aim of this chapter is to give the user the opportunity to develop new functions; in R, this corresponds to extending the language. The user can thus complete his comprehension of how R works.

SECTION 8.1

Preamble

The strength of the R system is that it includes a real programming language. We shall see that it offers very original programming concepts. The concept of objects is very present in R. Object-oriented programming as used in R is transparent for the user, in the sense that you do not need to understand the theory in order to use it. The same cannot be said for the developer who wishes to respect the spirit of R.

Practical Problem

As an example, this chapter will tackle the resolution of the following practical problem. Suppose that some users, beginners in R, wish to discover programming in R by developing a few functions relative to the well-known least squares methods,¹ in the context of simple linear regression. He soon realizes that two specific tasks

¹ See for example http://en.wikipedia.org/wiki/Ordinary_least_squares.

are of interest to him: first, output a summary with estimations and the coefficient of linear correlation; second, draw a scatter plot with the regression line. With his experience from previous chapters, this user finds it easy to produce these results from the command line. However, he/she would like to avoid having to type in several lines of commands every time he/she wishes to see the result of these two tasks, and so would like to develop two functions, easier to apply in a daily use of R. To this end, he/she will have the help of a more advanced user who can advise him/her every time he/she encounters a difficulty.

This practical problem should help the reader understand the use of the notions presented in this chapter.

SECTION 8.2

Developing Functions

First of all, let us introduce some basic theoretical elements to explain how to create a function in R.

8.2.1 Quick Start: Declaring, Creating and Calling Functions

Declaring a function is done with the following general form:

function(<list of arguments>) <body of the function>
where

- <list of arguments> is a list of named (formal) arguments.
- <body of the function> represents, as the name suggests, the contents of the code to execute when the function is called.

Here is an example of function declaration:

```
> function(name) cat("Hello",name,"!")  
function(name) cat("Hello",name,"!")
```

For R, a function is a specific object. Creating a function thus corresponds to affecting the object “R function” to a variable, the name of which corresponds to the function itself. For example, to create the function hello(), you can proceed as follows:

```
> hello <- function(name) cat("Hello",name,"!")  
> hello  
function(name) cat("Hello",name,"!")
```

For this function to be executed, the user needs to call the function, followed by the effective arguments listed in brackets. Recall that an **effective argument** is the

value affected to a formal argument. We will use the terms calling argument and input argument as synonyms of effective argument.

```
> hello("Peter")
Hello Peter !
```

8.2.2 Basic Concepts on Functions

8.2.2.1 Body of a Function

The body of a function can be a simple R instruction, or a sequence of R instructions. In the latter case, the instructions must be enclosed between the characters { and } to delimit the beginning and end of the body of the function. Several R instructions can be written on the same line as long as they are separated by the character ;. When the body of the function includes several R instructions written on the same line, do not forget to enclose them between characters { and }. Recall that on a line, any code written after the character # is not interpreted by R and is taken to be a comment.

```
> hello <- function(name) {
+   # Convert the name to upper case.
+   name <- toupper(name)
+   cat("Hello",name,"!")
+ }
> hello("Peter")
Hello PETER !
```

8.2.2.2 List of Formal and Effective Arguments

In this section, we describe how to declare the list of formal arguments when defining a function and how to input the list of effective arguments when calling a function.

Declaring a Function

When declaring a function, **all arguments are identified by a unique name**. Each argument can be associated with a default value. To specify a default value, use the character = followed by the default value, as when declaring a list object (`list()`). When the function is called with no effective argument for that argument, the default value will be used. We have used this functionality many times in previous chapters, but we now know how to include it when developing new functions. Here is an example:

```
> hello <- function(name="Peter") cat("Hello",name,"!")
> hello()
Hello Peter !
```

It seems useful to explain the difference between calling the name of the function `hello` and calling the function followed by brackets: `hello()`. The first form will display the contents of the function, as with any other **R** object, whereas the second form will call the function (in this case, with no argument specified). To execute a function, you always have to add brackets and list the effective arguments if necessary.

Naming Effective Arguments

In **R**, an effective argument can be entered by adding the name of the formal argument. Of course, this is of little interest when the function only depends on a single formal argument. Let us add to our function `hello()` the possibility of choosing a language, and see a few calls of this function.

```
> hello <- function(name="Peter",language="eng") {
+   cat(switch(language,fr="Bonjour",sp="Hola",eng="Hello"),
+        name,"!")
+ }
> hello()
Hello Peter !
> hello(name="Ben")
Hello Ben !
> hello(language="fr")
Bonjour Peter !
```

This functionality, combined with the ability to specify default values,² allows the developer to define a function with an important list of formal arguments corresponding to call options. Users can then call this function without needing to input all effective arguments. For example, they can affect a value to the last formal argument without having to type in all the other effective arguments. This way, a single function can be used for what would have otherwise required several functions. This is a true specificity³ of **R**, which allows an innovative programming mode. For example, read the help file on the functionalities of the function `seq()` with the various arguments `by`, `length.out` and `along.with`.

Partial Naming of Effective Arguments

In the same context, a second functionality of **R** is that it allows calling a function without typing in the complete name of a formal argument. Consider the following calls of the function `hello()`:

```
> hello(lang="eng")
Hello Peter !
> hello(l="eng")
Hello Peter !
> hello(l="e")
Peter !
```

² The function `missing()` is also very useful for this kind of programming.

³ It should be noted that many programming languages do not have this functionality.

The rule for determining the formal argument corresponding to a partial name is: in the ordered list of formal arguments of the function, the selected formal argument is the **first** formal argument for which there is a match between the first letters of the argument name and the partial name given by the user.

List of Supplementary Arguments “...”

You can give a list of supplementary arguments with the syntax `....`. When calling the function, all “named” arguments which are not in the list of formal arguments are grouped in the structure `....`. In the body of the function, the user can then use the syntax `...` as if copy-pasting the list of supplementary named arguments. This begs for an example:

```
> test.3points <- function(a="foo",...) print(list(a=a,...))
> test.3points("bar",b="foo")
$a
[1] "bar"
$b
[1] "foo"
```

Generally speaking, a rule of thumb for using the list of supplementary arguments `...` in the body of a function is that it should be used as an argument of one or several internal function calls.

Advanced users

When `...` is included in a list of arguments and is not in last position, “partial naming of arguments” will not work for all arguments after `....`. Indeed, a partial formal argument name is then considered as a formal argument in the supplementary list.

```
> test.3points <- function(aa="foo",...,bb="bar") {
+           print(list(aa=aa,...,bb=bb))}
> test.3points(a="bar",b="foo")
$aa
[1] "bar"
$b
[1] "foo"
$bb
[1] "bar"
```



Note that the value of the formal argument `aa` has been modified, but that `bb` did not change its value. The formal argument `b` was created. To change the value of the second formal argument `bb`, you need to use the complete name.

```
> test.3points(a="bar",bb="foo")
$aa
[1] "bar"
$bb
[1] "foo"
```

A keen user of partial names might be surprised by the following output when using the function `paste(..., sep = " ", collapse = NULL)` if he/she had taken the liberty of using the partial name (`col`) of the formal argument `collapse`:

```
> paste(c("foo", "bar"), col=" ", )
[1] "foo , " "bar , "
```

Since partial naming is ineffectual, `col` is considered as a second vector to `paste`, and the default options of the function `paste()` are used (i.e. `sep=" "` and `collapse=NULL`). To get the desired output, you need to use the complete name of the formal argument `collapse`.

```
> paste(c("foo", "bar"), collapse=" ")
[1] "foo, bar"
```

Tip

Generally speaking, when you call a function, you need to specify the value of all formal arguments for which no default value is defined. If you do not, an error occurs. There are however two exceptions. The first corresponds to the case where the argument is not used in the body of the function; this is of course useless and is probably due to a programming mistake. The second exception is when the developer allowed for this case in the body of the program, with the function `missing()`.



```
> hello <- function(name) {
+   if(missing("name")) name <- "Peter"
+   cat("Hello", name, "!")
+ }
> hello()
Hello Peter !
```

8.2.2.3 Object Returned by a Function

The function `hello()` above does not return any object. It simply produces a display on the screen.

```
> res <- hello()
Hello Peter !
> res
NULL
```

In previous chapters, we have often used R functions and saved the result as a variable (e.g., `x <- c(1, 5, 3)`), where the result of the base function `c()` is affected to the variable `x`). Since we are now interested in developing functions, let us examine how to create a function which returns an object (a result that is not ephemeral).

A general rule to return an object is to use the function `return()`. This instruction halts the execution of the code of the body of the function and returns the object between brackets. Here is an example:

```
> hello <- function(name="Peter") {  
+   return(paste("Hello",name,"!",collapse=" "))}  
> hello()  
[1] "Hello Peter !"  
> message <- hello()  
> message  
[1] "Hello Peter !"
```

The first call of the function returns the string of characters object without affecting it to a variable. The result is thus displayed on the screen, as if the user had entered in the command line the object returned by the function. The second call does not produce any display: the result of the function is redirected to the variable `message`, as the last instruction above shows.

Note

It is possible to return an object without using the function `return()`. The rule is then that the returned object is the last object manipulated in the last instruction of the body of the function (i.e. just before exiting the function). In the previous example, we could therefore have omitted the function `return()`

```
> hello <- function(name="Peter") {  
+   paste("Hello",name,"!",collapse=" ")  
> hello()  
[1] "Hello Peter !"
```



However, we discourage this practice because it does not always work, as shown below where we would expect that the function returns 10:

```
> function.without.return <- function() {  
+   for (i in 1:10) x <- i}  
> function.without.return()
```

Can you tell whether the following function returns an object? If yes, what is the content of this object?

```
> hello <- function(name="Peter") {  
+   msg <- paste("Hello",name,"!",collapse=" ")}
```

What do you think when you see the output below?

```
> hello()
```

There is no display, so it seems that no object is returned. But are you certain when you see the following example?

```
> message <- hello()  
> message  
[1] "Hello Peter !"
```

The last manipulated object is indeed the variable `msg`. Affecting the output to the variable `message` does store the contents of the variable `msg` from the body of the function. R can sometimes be unsettling, but you will agree that this kind of usage is not rational and a developer would probably never find it useful.

Tip

If you wish to get the same behaviour as in the last example, i.e. that the function does not display anything when called but does return an object, it is more direct to use the function `invisible()`—the name of this function is clear enough.



```
> hello <- function(name="Peter")
+ invisible(paste("Hello",name,"!",collapse=" "))
> hello()
> message <- hello()
> message
[1] "Hello Peter!"
```

8.2.2.4 Variable Scope in the Body of a Function

The notion of variable scope is very important for a language which allows to develop functions. The main point is that variables defined inside the body of a function have a local scope during function execution. This means that a variable inside the body of a function is physically different from another variable with the same name, but defined in the workspace of your R session. Generally speaking, local scope means that a variable only exists inside the body of the function. After the execution of the function, the variable is thus automatically deleted from the memory of the computer. We are now going to modify our function `hello()` by inserting controls of the contents of variables.

```
> message <- "hello Pierre !"
> message # Workspace initialization.
[1] "hello Pierre !"
> hello <- function(name="Peter",message="hello") {
+   print(message)
+   message <- paste(message,name,"!",collapse=" ")
+   print(message)
+   invisible(message)
+ }
> hello()
[1] "hello"
[1] "hello Peter !"
> message # Workspace has not been modified!
[1] "hello Pierre !"
> message <- hello()
[1] "hello"
[1] "hello Peter !"
```

```
> message # Workspace has been modified!
[1] "hello Peter !"
> message <- hello(message="Welcome")
[1] "Welcome"
[1] "Welcome Peter !"
> message # Workspace has been modified again!
[1] "Welcome Peter !"
```

A quick comment on the arguments of the function: contrary to what you might think, the variables `name` and `message` are not directly evaluated (initialized to the calling value or to the default value) before the execution of the body of the function. They are only initialized when they are first used in the body of the function. Recall that the function `missing()` is used to test whether a formal argument has been defined when calling the function. The only way for this functionality to be operational is by not evaluating the list of formal arguments at the beginning of the body of the function. Similarly, at the beginning of the body of the function, it is possible to get the effective call (with the completed list of arguments) by using the function `match.call()`.

```
> test.call <- function(aa="bar",...,bb="foo") {
+   print(match.call())
> test.call(a="foo",b="bar")
test.call(aa = "foo", b = "bar")
```

Advanced users

The last function creation may not seem very useful, but once you are an advanced R developer, you might find a use to the result of the function `match.call()`. We shall not give details, but only a taste of what can be done in R. We shall modify the last function so that it returns the arguments split into two lists: one (called `function`) of effective arguments associated with formal arguments and one (called `misc`) of supplementary effective arguments. Note how partial naming of arguments is managed.

```
> test.call <- function(aa="bar",...,bb="foo") {
+   args <- as.list(match.call())[-1]
+   inside <- names(args) %in% names(list(...))
+   list(func= args[!inside],misc=args[inside])
+ }
> test.call(a="foo",b="bar")
$func
$func$aa
[1] "foo"
$misc
$misc$b
[1] "bar"
```



A few lines of code are enough to get the result: introspection is easy in R and has many other features in the same context. We are not trying to get you to delve straight away into this kind of development, but wish to point out the possibilities of the language.

8.2.3 Application to the Practical Problem

After these theoretical explanations, our beginner user tries the following function codes for simple linear regression.

```

1 mysummary.reg1 <- function(y,x) {
2   aEst <- cov(x,y)/var(x)
3   bEst <- mean(y)-aEst*mean(x)
4   return(list(aEst=aEst, bEst=bEst, cor=cor(x,y)))
5 }
6
7 mydisplay.reg1 <- function(y,x) {
8   aEst <- cov(x,y)/var(x)
9   bEst <- mean(y)-aEst*mean(x)
10  plot(x,y)
11  abline(a=bEst, b=aEst)
12 }
```

Note

 Note that in old versions of R, you could write
`return(aEst=aEst, bEst=bEst, cor=cor(x,y))`
but that this usage will be deprecated in future versions.

After loading these functions with a copy–paste or with the command `source()`, the user tests an uninteresting example.

```

> y <- rnorm(10);x <- 1:10
> mysummary.reg1(y,x)
$aEst
[1] -0.1019453
$bEst
[1] 0.7822879
$cor
[1] -0.4198245
```

The instruction `mydisplay.reg1(y,x)` produces Fig. 8.1 on page 211.
We shall see later on how these functions can be enriched.

8.2.4 Operators

Calling a function under the form `<function>(<list of call arguments>)` is not always easy. An example is the function `seq()`. Of these two equivalent forms, which one do you prefer?

```

> seq(1,3)
[1] 1 2 3
```

```
> 1:3
[1] 1 2 3
```

You probably prefer the second form, since it is more synthetic (no brackets) and is thus easier to manipulate, for example, when using indices (of vectors, matrices, etc.). This form corresponds to an operator. R uses operators internally.

There are two forms of operators:

- **Unary operator** (one argument) : <operator> <argument1>
- **Binary operator** (two arguments) : <argument1> <operator> <argument2>

where <operator> is the operator, and <argument1> and <argument2> are the effective arguments of the operator. Here is a partial list of operators used internally by R:

```
+, -, *, /, ^, %%, %/%, &, |, !, ==, !=, <, <=, >, >.
```

A priori, these operators cannot be modified by the user.⁴ It is however possible to define extra operators. They are of the form %<operator>% and some are already available in the base system, for example, %in% and %o% (seen in Chap. 5).

Tip

To display the source of the function (the operator) %in%, use the instruction `get("%in%")`. You can see that it uses the function `match()` which you may find useful.



Suppose we wish a more synthetic way to concatenate strings of characters, which is normally done with the function `paste()`.

```
> "%+" <- function(ch1,ch2) paste(ch1,ch2,sep="")
> name <- "Peter"
> "The life of " %+% name %+% " is beautiful!"
[1] "The life of Peter is beautiful!"
> # This is a simplification of:
> paste("The life of ", name , " is beautiful!",sep="")
[1] "The life of Peter is beautiful!"
```

Note that since the name of the function is not alphanumeric, it has to be put between quotation marks. It is of course up to you whether you prefer one or the other form. We are not trying to diminish the usefulness of the function `paste()`, which is a much richer function than the simple operator %+% we have created (the creation actually used the function `paste()`). We are rather trying to show the flexibility of R which allows, with a simple function definition, a simplification of the calling syntax.

⁴ In fact, this group of operators can be used by a user when developing a new class of objects. But this matter is too advanced for this book!

Tip

You can use operators to define operations on sets, such as those presented on p. 99. For example, the union between two sets A and B can be defined as



```
> "%union%" <- function(A,B) union(A,B)
> A %union% B
[1] 4 6 2 7 1 3
```

8.2.5 R Seen as a Functional Language

R is a functional language in the sense that almost any code execution in R is done by calling functions, possibly scattered with control structures. In fact, you may be surprised to learn that the following features of R are also controlled by functions. We have seen that simply calling an R object results in the display of its contents. In fact, in such an instruction, R calls (without notifying the user) the function `print()` with effective argument the name of the object. Because this function is often used in R, it has a particular status; we shall discuss this further later on. All affectation operations (i.e. instructions with `<-`) are handled by functions whose names include (no surprise here) the distinctive sign `<-5`. Developing and maintaining the R system can be summarized as the construction of a range of functions. First are the basic functions, included in the basic installation of R. Usually, they cannot be modified by the user⁶, and even when they can be, we strongly advise against it; let your system become unusable. Second are the functions developed directly in R⁷ by any user. Many functions are made available by the community of R developers through a system of packages (more on this later).

SECTION 8.3

† Object-Oriented Programming

In this section, we shall view an object as more than a quantity that can be saved and reused. We shall come closer to the spirit of the R language by looking at the internal object-oriented mechanism which governs most of its use. The incredible part is that the user does not need to worry about knowing the internal workings of R. According to us, this is a strong point of R. Nonetheless, this section should

⁵ To see this, type in the command line `apropos("<-")`.

⁶ The core of R is developed in the C language for obvious reasons of speed of execution, which makes it rather reactive when used in the command line.

⁷ To speed up execution, it is usually possible to convert an R function into C and then to call it from R via the C API.

help users better understand how R proposes results. We expect this will lead to a less “random” and more controlled use of R.

8.3.1 How the Internal Object-Oriented Mechanism Works

8.3.1.1 Class of an Object and Declaring an Object

What matters in R is specifying the class of an object with the function "class<-"(). Recall that the function `class()` is used to check the class of an object.

```
> obj <- 1:10
> class(obj)
[1] "integer"
> class(obj) <- "MyClass"
> class(obj)
[1] "MyClass"
> class(obj) <- "OtherClass"
> obj
[1] 1 2 3 4 5 6 7 8 9 10
attr(,"class")
[1] "OtherClass"
```

The object `obj` of class `integer` is now an object of class `OtherClass`. The last display of the object `obj` indicates the class of the object, where `attr` stands for attribute. We shall come back to the notion of attributes at the end of this chapter. For now, it is enough to understand the meaning of the display `attr(,"class")` which is literally the “class attribute”.

Advanced users

That said, the above is not quite true: the object `obj` has kept the characteristic of also being of the `integer` class, as the following output shows:

```
> obj*2
[1] 2 4 6 8 10 12 14 16 18 20
attr(,"class")
[1] "OtherClass"
```

Indeed, all the elements of the vector `obj` have been multiplied by 2. We hope that in future versions of R, the output of the function `class()` applied to such an object will be similar to [1] "OtherClass" "integer", which would better show the true nature of the object.

There are two ways of knowing whether an object is of a given class:

```
> class(obj) == "MyClass"
[1] FALSE
```



```
> inherits(obj, "MyClass")
[1] FALSE
```

The function `inherits()` should be preferred, as we shall see when we consider polymorphic objects with several classes.

Tip

To see the class of the function `function()`, you can use this instruction:



```
> class(function() {})
[1] "function"
```

For the function ":"(), use `class(get(":"))`.

8.3.1.2 Declaring Objects and Using Methods

The mechanism for object-oriented programming is rather simple and original in R, compared to many other languages. To illustrate this mechanism, examine the most used example in R: the display of an object with the function `print()`. Examine the following R outputs:

```
> vect <- 1:10
> class(vect)
[1] "integer"
> vect
[1] 1 2 3 4 5 6 7 8 9 10
> print(vect)
[1] 1 2 3 4 5 6 7 8 9 10
```

No surprises so far, although it is worth pointing out that simply entering an R object in the command line seems to provoke a call to the function `print()` with the given object as effective argument. The next example confirms this idea⁸: it displays an object of the class `formula`, characterized by the tilde symbol (~). In this example, we save in the variable `form` the formula expressing the relationship between `y` and `x`. Note that the objects `y` and `x` do not need to exist, since no evaluation is done when a formula is defined.⁹

```
> form <- y~x
> class(form)
[1] "formula"
> form
y ~ x
> print(form)
y ~ x
```

⁸ In fact, for auto-printing base objects (vectors, matrices, lists, etc.) in the console, R does not use the `print()` function, but calls a C function named `PrintValueEnv`, which is not directly available to the user.

⁹ No further details are needed for now; we shall come back to this very original class of objects.

Note that the function `print()` works differently for different classes of objects. For the variable `form` (of class "formula"), `print()` returned `y~x`, which is the instruction to the right of the affectionate arrow. For the variable `vect`, calling `print()` returns `[1] 1 2 3 4 5 6 7 8 9 10` when we might have expected it to display `1:10`. Here is the code of the function `print()`:

```
> print
function (x, ...)
UseMethod("print")
<environment: namespace:base>
```

The body of this function indicates that the function `UseMethod()` must be executed. This function is a *generic function* in R. Like an airport traffic control tower, it is used to redirect the object, according to its class, to the correct function call. In the last example, this corresponds to calling the display function associated with the class `formula` of the form `print.formula()`. In the object-oriented programming vernacular, such functions, of the general type `<method>. <class>`, are called *methods*. This explains the name of the function `UseMethod()` in the body of the generic function `print()`.

Here is what happens in the backstage to simply display the object `form`:

```
> form # Calls the function print(),
      # which calls the function print.formula().
y~x
> print.formula(form)
y~x
```

Advanced users

To check how easy it is to change the general behaviour of R by changing one function, we are going to redefine the display function for the class `formula`. We are simply going to keep the standard display and add the string of characters "`formula:`".

```
> print.formula <- function(obj,...) {
+ cat(paste("formula:",paste(sapply(obj[c(2,1,3)],
+ as.character),collapse="")))
+ invisible(obj)
+ }
> y~x
formula: y~x
```

If you are a beginner in R, you should not try to understand the details of the R code leading to this result. Although the code seems simple, understanding it requires notions which we cannot go into in this book. Once again, the aim is rather to reveal the introspective power of R, since even its base elements can be manipulated.

To restore the initial behaviour of R for displaying formulae, you will have guessed that it suffices to delete the new function `print.formula()` with the command line instruction `rm(print.formula)`. We shall not delete it yet, because we need this behaviour later on.



If you have understood the way the function `print()` works, you might expect that there exists a function `print.integer()`. We can check this:

```
> print(vect)
[1] 1 2 3 4 5 6 7 8 9 10
> print.integer(vect)
Error in eval(substitute(expr), envir, enclos) :
  could not find function "print.integer"
```

The function `print.integer()` does not exist. In fact, when there is no method associated with a class, R executes the default method, which is of general form `<method>.default`; in this case, `print.default()`. Here is the output of this function for our two examples:

```
> print.default(vect)
[1] 1 2 3 4 5 6 7 8 9 10
> print.default(form)
y ~ x
attr(,"class")
[1] "formula"
attr(,".Environment")
<environment: R_GlobalEnv>
> # Compare with:
> form
formula: y~x
```

We now have a complete explanation of what happens behind the scenes. We also see that the display of a formula does not use the default method, as the last output suggests.

Tip

Also note that the function `print.default()` is used to display all base objects (or structures) of R when these objects are taken as effective arguments of the function `print()`.

In summary, to define a new family of methods, denoted here by `<method>` (name of the family of methods you wish to create), which can be applied to any type of object, you need to:

- First declare the ***generic function*** in the following form:
`<method> <- function(obj,...) UseMethod("<method>")`
- Then create a ***method*** `<method>` for a class `<class>`:
`<method>.<class> <- function(obj,<list of arguments>) <body of the method>`
 where `<list of arguments>` and `<body of the method>` are, respectively, an optional list of formal arguments and the contents of this method, which is nothing else than a function when called in its long version.

Note

Note that when declaring a family of methods, you can dissociate the name of the generic function and the argument of the function `UseMethod()` corresponding to the name of the method to call. Thus, it is easy to define an alias, called `<alias>`, of the last family of methods by simply defining a new generic function:

```
<alias> <- function(obj,...) UseMethod("<method>")
```

As a result, the two command line calls `<method>(<object>)` and `<alias>(<object>)` for an object `<object>` of class `<class>` are equivalent to `<method>.(<class>)(<object>)`. A rather surprising application is that a method can be translated like this. In the next example, the French *voir* is used as an alias of `print`:



```
> voir <- function(obj,...) UseMethod("print")
> voir(vect)
[1] 1 2 3 4 5 6 7 8 9 10
> voir(form)
formula: y~x
> rm(print.formula) # Remove our method to return
# to the normal mode.
> voir(form)
y ~ x
> form
y ~ x
```

8.3.2 Back to the Practical Problem

The user realizes that he/she has repeated the execution of the estimations of `a` and `b` twice when creating the functions `mydisplay.reg1()` and `mysummary.reg1()` introduced in Sect. 8.2.3 (lines 2 and 3, and lines 8 and 9). He asks advice from a more advanced user, who suggests using the concept of object-oriented programming. He/she proposes to create a function¹⁰ to return an object of class `reg1`, so that it can be reused thereafter as first calling argument for any method of the said class.

```
1 reglin <- function(y,x) {
2   aEst <- cov(x,y)/var(x)
3   bEst <- mean(y)-aEst*mean(x)
4   reg <- list(y=y,x=x,aEst=aEst,bEst=bEst)
5   class(reg) <- "reg1"
6   return(reg)
7 }
```

¹⁰ This kind of function is often called a constructor in object-oriented programming.

They now define the method `mydisplay.reg1()` which can be used on any object of class `reg1`.

```

1 mydisplay.reg1 <- function(reg) {
2   plot(reg$y, reg$x)
3   abline(a=reg$bEst, b=reg$aEst)
4 }
5
6 mysummary.reg1 <- function(reg) return(reg)
```

They try a few tests:

```

> reg <- reglin(y,x)
> mysummary(reg)
Error in eval(substitute(expr), envir, enclos) :
  could not find function "mysummary"
> mydisplay(reg)
Error in eval(substitute(expr), envir, enclos) :
  could not find function "mydisplay"
```

The user did not expect such errors, so he/she checks that the function is well defined:

```

> mysummary.reg1(reg)
$y
[1] 1.8920106 0.3978771 -0.3970281 -0.2799578 0.7851185
[6] -0.2103208 0.1921150 -0.2647256 -0.5013911 0.6021898
$x
[1] 1 2 3 4 5 6 7 8 9 10
$aEst
[1] -0.1019453
$bEst
[1] 0.7822879
attr(,"class")
[1] "reg1"
```

The advanced user points out the mistake: the generic functions `mysummary` and `mydisplay` have not been declared and are not standard, unlike a few others such as `print()` and `summary()`.

```

1 mysummary <- function(x,...) UseMethod("mysummary")
2 mydisplay <- function(x,...) UseMethod("mydisplay")
```

The previous instructions now work:

```

> mysummary(reg)
$y
[1] 1.8920106 0.3978771 -0.3970281 -0.2799578 0.7851185
[6] -0.2103208 0.1921150 -0.2647256 -0.5013911 0.6021898
$x
[1] 1 2 3 4 5 6 7 8 9 10
$aEst
[1] -0.1019453
```

```
$bEst
[1] 0.7822879
attr(),"class")
[1] "reg1"
> mydisplay(reg)
```

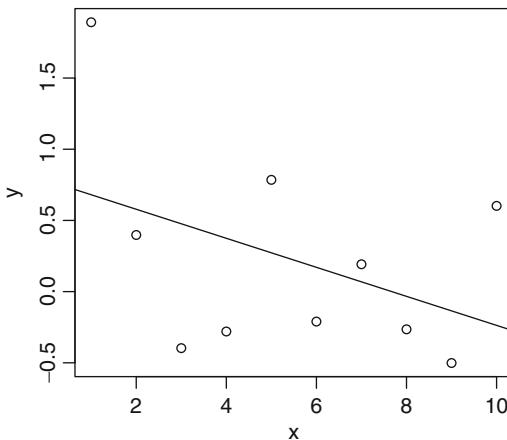


Fig. 8.1: Result of the call of the function `mydisplay.reg1()`

Since the method `print.reg1()` has not been defined, you may wonder what would happen when we simply enter the name of the object.

```
> reg
$y
[1] 1.8920106 0.3978771 -0.3970281 -0.2799578 0.7851185
[6] -0.2103208 0.1921150 -0.2647256 -0.5013911 0.6021898
$x
[1] 1 2 3 4 5 6 7 8 9 10
$aEst
[1] -0.1019453
$bEst
[1] 0.7822879
attr(),"class")
[1] "reg1"
```

We already knew that the method `print.default()` is called in such cases.

8.3.3 *Information About Methods*

To get information about methods, R has the function `methods()`:

```
> methods("formula") # Or more directly methods(formula).
[1] formula.character*   formula.data.frame*  formula.default*
[4] formula.formula*    formula.glm*        formula.lm*
[7] formula.nls*        formula.terms*
```

```

Non-visible functions are asterisked
> methods(class="formula")
[1] [.formula*
[3] alias.formula*
[5] ansari.test.formula*
[7] boxplot.formula*
[9] cor.test.formula*
[11] deriv3.formula
[13] formula.formula*
[15] ftable.formula*
[17] kruskal.test.formula*
[19] mood.test.formula*
[21] pairs.formula*
[23] points.formula*
[25] prcomp.formula*
[27] print.formula
[29] selfStart.formula*
[31] stripchart.formula*
[33] terms.formula
[35] var.test.formula*
Non-visible functions are asterisked

```

Warning

 Do not confuse the two uses. The first instruction outputs all methods (of the form `<method>.formula`) associated with the generic function `formula`. The second instruction gives all methods for the class `formula`.

Here are a few examples to better understand the distinction between the two uses of the function `methods()`.

```

> class(y~x)
[1] "formula"
> update(y~x,.~.+z) # Apply the method update() to an
                      # object of class formula.
y ~ x + z
> update.formula
function (old, new, ...)
{
  tmp <- .Internal(update.formula(as.formula(old),
                                    as.formula(new)))
  out <- formula(terms.formula(tmp, simplify = TRUE))
  return(out)
}
<environment: namespace:stats>
> form <- "y~x"
> class(form)
[1] "character"
> formula(form)
y ~ x
> formula.character
Error: object "formula.character" not found

```

Tip

Functions followed with an asterisk can be executed, but the body of the function cannot be visualized. You can however use the function `getAnywhere()`.

```
> getAnywhere(formula.character)
A single object matching 'formula.character' was found
It was found in the following places
  registered S3 method for formula from namespace stats
  namespace:stats
with value
function (x, env = parent.frame(), ...)
{
  ff <- formula(eval(parse(text = x) [[1L]]))
  environment(ff) <- env
  ff
}
<environment: namespace:stats>
```



8.3.4 Inheriting Classes

In the context of our practical problem, the advanced user informs the beginner user that R already has a set of functions to manage linear models. Indeed, the function `lm()` is dedicated to this kind of treatment (as we shall see in Chap. 14). However, he/she adds that to his knowledge, no functions exist to perform the specific treatment they propose. The two users work together to develop an extension; they want to avoid “reinventing the wheel” and make the most of existing functions in R.

In object-oriented programming, the notion of class inheritance seems appropriate for this kind of extension. Inheritance expresses the fact that an object of a certain class can also behave like all objects of supplementary classes. Such a mechanism is available in R, by associating a sequence of classes with an object. Thus, when a method is applied to an object which has a hierarchy of classes, the first class is solicited first. If the method exists for this class, it is executed. Otherwise, R tests whether there is an executable method in the class hierarchy. If there is, that method is executed; otherwise, the default method is executed, as long as it is defined. Finally, if none of the above apply, an execution error is generated. Let us illustrate this notion with the problem of our two users. First, we need to declare the constructor function of the new class `lm1`, which inherits directly from the existing class `lm`.

```

1 lm1 <- function(...) {
2   obj <- lm(...)
3   if(ncol(model.frame(obj))>2) stop("more than one
4     independent variable")
5   class(obj) <- c("lm1", class(obj)) # Or c("lm1","lm")
6   obj
7 }
```

Apply this to the same variables as before.

```

> reg <- lm1(y~x)
> reg
Call:
lm(formula = . . .)
Coefficients:
(Intercept)           x
0.7823            -0.1019
```

We can see inheritance in action. No method `print.lm1()` is defined, and yet the object is not displayed as with `print.default()`. This is because R already knows the method `print.lm()` and the object `reg` inherits methods from the class `lm`. There are several ways of checking that this object is indeed inheriting from this class; the simplest is visualizing the contents of the `class` attribute with the function `class()`. A developer might prefer the more direct function `inherits()`.

```

> class(reg)
[1] "lm1" "lm"
> inherits(reg, "lm")
[1] TRUE
> print.lm(reg)
Call:
lm(formula = . . .)
Coefficients:
(Intercept)           x
0.7823            -0.1019
```

Line 3 (which we shall not comment) in function `lm1()` tests whether the formula is a simple regression model formula. See what happens in this next example:

```

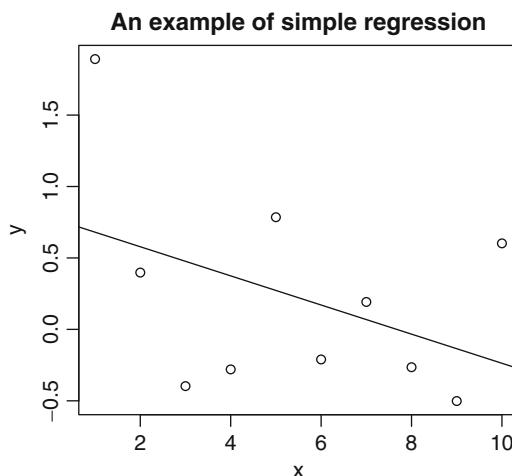
> lm1(y~x+log(x))
Error in lm1(y ~ x + log(x)) : more than one
      independent variable
```

We continue developing functions in the same spirit as

```

1 plot.lm1 <- function(obj, ...) {
2   plot(formula(obj), ...)
3   abline(obj)
4 }
```

```
> summary(reg)
Call:
lm(formula = ...1)
Residuals:
    Min      1Q  Median      3Q     Max 
-0.8735 -0.3772 -0.2060  0.4153  1.2117 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 0.78229   0.48348   1.618   0.144    
x           -0.10195   0.07792  -1.308   0.227    
Residual standard error: 0.7077 on 8 degrees of freedom 
Multiple R-squared:  0.1763,    Adjusted R-squared:  0.07328 
F-statistic: 1.712 on 1 and 8 DF,  p-value: 0.2271 
> plot(reg,main="An example of simple regression")
```



In the call of `summary()` above, the method `summary.lm1()` has not been developed; hence, the standard method `summary.lm()` is executed. Indeed, the object `reg` of class `lm1` then inherits from the class `lm` for all standard methods proposed by **R** to manage linear models. For the call of the method `plot()`, the freshly created method `plot.lm1` is invoked.

Note

Note that **R** has a standard method `plot.lm()` which creates a set of plots for a more detailed analysis of the results (see Chap. 14). We have intentionally changed the default behaviour of **R** for simple linear regression, but can still access this method by calling it explicitly (`plot.lm(reg)`).



Advanced users

Object-oriented programming is extremely simple in its conception. There are many object-oriented programming languages. An important difference is that the vast majority offer an encapsulation of object fields and methods; one of the points of this encapsulation is that the fields of an object can be modified within a method. This is not directly possible in R because of the strict local scope of variables inside the code of an R function. The users can however adopt this kind of programming if they want to. Any method `<method>.<class>()` which needs to modify the fields of an object `<object>` (of class `<class>`) must then return the object itself. The user of the generic function `<method>()` can then affect the result to the initial object, as follows:

 `<object> <- <method>(<object>)`. However, this risks to slow down execution, all the more if the contents of the object fields are large. This is because the object is completely duplicated. We hope that R developers will one day offer a more elegant standard functionality (analogous to what the majority of object-oriented programming languages offer), whereby only the relevant fields (of which there are usually few) are modified inside the body of the method. When you become an advanced user (as we hope), you will notice that the notion of pointers (which is very common in programming) is not directly offered to R developers (see however the function `tracemem()` as well as Sect. 9.8.2.2, p. 296).

SECTION 8.4**† Going Further in R Programming**

Before you start programming in a language, it is good to know the spirit in which it was conceived. In this section, we shall explore structures of the R language which you do not need to know when you start using R, but which you will find very useful when you decide to go deeper in your use of R. These elements make R an original and powerful tool. We advise beginner users to skim through this section without trying to master the concepts. All the information in this section is second level, in the sense that a very powerful use of R is possible without it.

8.4.1 R Attributes

An R object includes *primary information*, conveyed by the basic structures presented in this book. There is another level of information, which we call *secondary information*. It is attached to an object with attributes and can be accessed with the function `attributes()`.

```
> mat <- matrix(1:10,nrow=2)
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> class(mat)
[1] "matrix"
> attributes(mat)
$dim
[1] 2 5
```

We shall comment on this output later. For now, let us insist again on the fact that this mechanism is supposed to be transparent for the user, who usually cares more about the contents of the R object. For day-to-day use, we advise you not to change attributes directly. This stand is justified by the existence of many functions to manipulate attributes indirectly. However, a developer who wishes to learn more about the internal workings of R will discover a few supplementary characteristics which usually enlighten the behaviour of the object. We have already indirectly manipulated the attribute `class` with the functions `class()` and "`class<-()`". We shall also manipulate the three other main attributes: `dim`, `names` and `dimnames`. These are used a lot in the internal management of R. The next example is only interesting to present how to handle attributes. The complementary function `attr()` is used to manipulate a single attribute at a time, whereas the function `attributes()` returns all attributes as an R list.

```
> vect <- 1:10
> attr(vect,"test") # Returns NULL, because vect has no
                     # attribute test.
NULL
> attributes(vect) # NULL because vect has no attributes.
NULL
> # Affecting an attribute "attrib1" containing the character
# string "TEST1".
> attr(vect,"attrib1") <- "TEST1"
> attr(vect,"attrib1")
[1] "TEST1"
> # Affecting an attribute "attrib2" containing the vector c(1,3)
> attributes(vect)$attrib2 <- c(1,3)
> attributes(vect)
$attrib1
[1] "TEST1"
$attrib2
[1] 1 3
> attr(vect,"attrib2")
[1] 1 3
> # Modifying attribute "attrib1" and deleting attribute
# "attrib2"
> attributes(vect)$attrib1 <- 3:1
> attr(vect,"attrib2") <- NULL
> attributes(vect)
$attrib1
[1] 3 2 1
```

```
> # Deleting all attributes at once
> attributes(vect) <- NULL
> attributes(vect)
NULL
```

The attribute access mechanism is simple to use. This example has shown how to change attributes using the functions "attr<-"() and "attributes<-"(). The value of an attribute can be any R object. Affecting NULL to an attribute deletes it.

8.4.1.1 Attribute class

In Sect. 8.3, we have manipulated the attribute `class` using the functions `class()` and "class<-"(). This shows that you do not need to know how to manipulate attributes directly. We return to the example we used, to show that manipulating this attribute is equivalent to using the utility functions `class()` and "class<-"().

```
> form <- y~x
> attributes(form)
$class
[1] "formula"
$.Environment
<environment: R_GlobalEnv>
> class(form)
[1] "formula"
> obj <- 1:10
> attr(obj,"class") # No class attribute.
NULL
> class(obj)          # And yet!
[1] "integer"
> attr(obj,"class") <- "MyClass" # Equivalent to class(obj) <-
                                         # "MyClass".
> class(obj)
[1] "MyClass"
```

There is nothing left to say about this attribute, even though it plays a central role in object-oriented programming in R.

8.4.1.2 Attribute dim

The attribute `dim` plays an important role in the behaviour of `matrix` and `array` objects. Here is an example with a matrix:

```
> mat <- matrix(1:12,nrow=2)
> mat
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> attr(mat,"dim")
[1] 2 6
```

```

> attributes(mat)
$dim
[1] 2 6
> attr(mat,"dim") <- c(3,4) # Changing shape: 3 rows and 4
# columns.
> mat
[,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> attributes(mat)$dim <- c(2,6) # Back to the initial shape.
> mat
[,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12

```

In this example, changing the attribute `dim` allowed us to change the shape of the matrix. We have already mentioned that attribute management is meant to be transparent for the user, so you might expect there exist similar functions with more user-friendly names. For this example, we could have used the functions `dim()` and `"dim<-"`:

```

> dim(mat)
[1] 2 6
> dim(mat) <- c(1,12) # Changing shape: 1 row and 12 columns.
> mat
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
[1,]    1    2    3    4    5    6    7    8    9   10   11
[1,]    12
> dim(mat) <- c(2,6) # Back to the initial shape.

```

To really understand how R represents objects such as matrices and arrays, let us analyse the following output:

```

> mat
[,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> class(mat)
[1] "matrix"
> dim(mat) <- NULL # Or attributes(mat)$dim<-NULL or
# attributes(mat) <- NULL.
> mat
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> is.vector(mat)
[1] TRUE
> class(mat)
[1] "integer"
> dim(mat) <- c(2,2,3)
> mat
, , 1
[,1] [,2]
[1,]    1    3
[2,]    2    4

```

```

, , 2
[,1] [,2]
[1,]    5    7
[2,]    6    8
, , 3
[,1] [,2]
[1,]    9   11
[2,]   10   12
> is.vector(mat)
[1] FALSE
> class(mat)
[1] "array"

```

When we delete the attribute `dim`, the object `mat` becomes a simple vector. When we affect a vector of three integers to this attribute, the object `mat` becomes an array of dimension 3. The different behaviours of vectors, matrices and arrays thus stem from the value of the attribute `dim`.

Warning

Although the display is the same, a vector and a single-index array are treated differently by R, as shown by these few lines of code:



```

> dim(mat) <- 12
> mat
[1]  1  2  3  4  5  6  7  8  9 10 11 12
> is.vector(mat)
[1] FALSE
> class(mat)
[1] "array"
> identical(mat,1:12)
[1] FALSE
> dim(mat) <- NULL
> mat
[1]  1  2  3  4  5  6  7  8  9 10 11 12
> is.vector(mat)
[1] TRUE
> class(mat)
[1] "integer"
> identical(mat,1:12)
[1] TRUE

```

It looks like we have said everything about the attribute `dim`, but there is one last application worth noting. The only difference between a vector and a list is that the elements of a vector must all have the same type. Matrices and arrays usually contain elements of the same nature as well; this constraint is very important for matrix operations. But as storage structures, you could imagine extending the matrix and array concepts to lists, by affecting the `dim` attribute, as is done with vectors. The documentation files for the `matrix()` and `array()` instructions show that this is the case, since the first calling argument of these functions can be a list instead of

a vector. The next example applies this to a matrix; the same could be done with an array, as long as the number of elements in the list agrees with the dimension.

```
> lmat <- matrix(list(7,1:2,1:3,1:4,1:5,1:6),nrow=2)
> lmat          # Returns the structure and not the contents, which
   # are too difficult to display.
     [,1]      [,2]      [,3]
[1,] 7           Integer,3 Integer,5
[2,] Integer,2 Integer,4 Integer,6
> dim(lmat)
[1] 2 3
> is.list(lmat)
[1] TRUE
> lmat[1,2] # Extract the element at row 1 and column 2.
[[1]]
[1] 1 2 3
> lmat[,-2] # Extract the submatrix with the second column
   # removed.
     [,1]      [,2]
[1,] 7           Integer,5
[2,] Integer,2 Integer,6
> dim(lmat) <- NULL
> lmat          # This is just a list now.
[[1]]
[1] 7
[[2]]
[1] 1 2
[[3]]
[1] 1 2 3
[[4]]
[1] 1 2 3 4
[[5]]
[1] 1 2 3 4 5
[[6]]
[1] 1 2 3 4 5 6
> is.list(lmat)
[1] TRUE
```

8.4.1.3 Attributes `names` and `dimnames`

The attribute `names` plays an important role in naming elements of a list.

```
> li <- list(1:3,letters[1:3])
> li
[[1]]
[1] 1 2 3
[[2]]
[1] "a" "b" "c"
> attributes(li)
NULL
> attributes(li)$names <- c("numbers","letters")
> li
$numbers
```

```
[1] 1 2 3
$letters
[1] "a" "b" "c"
```

The first and fourth instructions are thus equivalent to the following, more common declaration:

```
> li <- list(numbers=1:3,letters=letters[1:3]))
```

It is a less useful and lesser known fact that this attribute can also be used on any type of vector.

```
> vect <- 1:3
> attr(vect,"names") <- letters[1:3]
> vect
a b c
1 2 3
> # Or directly
> vect2 <- c(a=1,b=2,c=3)
> vect2
a b c
1 2 3
```

You do not need to manipulate the attribute `names` directly. Accessing and changing its value can be done explicitly:

```
> names(li)
[1] "numbers" "letters"
> names(li) <- c("num","lett")
> li
$num
[1] 1 2 3
$lett
[1] "a" "b" "c"
> names(vect)
[1] "a" "b" "c"
> names(vect) <- toupper(names(vect))
> vect
A B C
1 2 3
```

For objects with several indices, such as matrices and arrays, index name management is done internally by modifying the attribute `dimnames`, as shown in this quick example.

```
> mat <- matrix(1:6,nr=2)
> mat
[,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> attributes(mat) # Can be modified as an attribute.
$dim
[1] 2 3
> rownames(mat)   # Row names.
NULL
> colnames(mat)   # Column names.
```

```

NULL
> dimnames(mat)    # Row and column names as a list.
NULL
> colnames(mat)   <- paste("V",1:3,sep="")
> rownames(mat)   <- c("a","b")
> mat
  V1 V2 V3
a  1  3  5
b  2  4  6

```

For an array with more than two dimensions, the functions `rownames` and `colnames` are meaningless. You can either modify the attribute `dimnames` directly or use the function `"dimnames<-"()`.

Note

Data frames have a special status. They are defined as lists and are usually manipulated as matrices. The attributes for row and column names are `row.names` and `names` (instead of `col.names`):

```

> df <- data.frame(a=1,b=1:2)
> df
  a b
1 1 1
2 1 2
> attributes(df)
$names
[1] "a" "b"
$row.names
[1] 1 2
$class
[1] "data.frame"
> names(df)      # As a list.
[1] "a" "b"
> dimnames(df)   # As an array: list of two vectors.
[[1]]
[1] "1" "2"
[[2]]
[1] "a" "b"
> rownames(df)   # As a matrix: accessing the row names.
[1] "1" "2"
> colnames(df)   # As a matrix: accessing the column names.
[1] "a" "b"

```



The last four lines give calls to access these attributes without manipulating them directly. Corresponding forms exist to change their values. Note that the attribute `class()` gives the class of the object.

8.4.2 Other R Objects

It could be said that one of the specificities of R is that the vast majority of quantities manipulated by R are allocated to variables and can thus be reused later on. There are a few exceptions, mostly control structures. R objects are of different types, called classes. We have already encountered object classes used to store common data. There are three other object types we chose to explore as well. Surprisingly, formulae and environments are also objects in R; we shall also introduce R expressions, which are objects in which R code can be stored to be executed at a later time.

8.4.2.1 R Expressions

So far, we have said nothing on structures used to described the syntactic bases of R. Following its philosophy of managing as many components as possible, R can manipulate an R expression and split it into a sequence of atomic entities (such as `call`, `name`...). We only mention these capacities, without going into the details. We shall focus on R expressions which are truly of interest to an R developer. It is difficult to give a rigorous definition of R expressions. We propose the following definition, inspired by command line use of R. An R expression can be seen as R code entered in sequence as command lines until it is executed by the R interpreter (i.e. until the character > is displayed, inviting a new command). This expression can spread over several lines. The function `expression()` is used to declare an R expression when it is used with a single calling argument. It is however possible to give a sequence of expressions, each expression corresponding to one effective argument in the call of a function. An expression object is not evaluated by the R interpreter but can be saved to be evaluated later, as many times as needed. Evaluating an R expression is done with the function `eval()`. All of this is illustrated in this example:

A developer will find it useful to convert a character string describing R code into an R expression to be evaluated at another time. The function `parse()` is used to this effect:

```
> parse(text='v<- "value"') -> expr
> expr
expression(v<- "value")
attr(,"srcfile")
<text>
> eval(expr)
> v
[1] "value"
```

The formal argument `text` is used here to read a character string, but the first use of the function is to read a file containing R code; the name of the file is given as the first effective argument.

Tip

Here is an example using the functions `eval()` and `parse()`:

```
> for (i in 1:3) eval(parse(text=paste("a",i," <- i",sep="")))
> a2
[1] 2
```



We are now going to manipulate the function `expression()` to describe some of the internal behaviour of R. This will help understand why R is said to be a functional language (i.e. which makes an intensive use of functions). It is surprising how true this is. This first point shows that upon execution, affectation is considered as an operator (a function with two arguments). The first argument corresponds to the variable, the second to the contents.

```
> foo <- "foo"
> foo
[1] "foo"
> "<-"(foo,"foo2") # Equivalent to: foo <- "foo2"
> foo
[1] "foo2"
> expression("<-"(foo,"foo2")) # as shown by the output of this
# expression.
expression(foo <- "foo2")
```

We continue our exploration with brackets. One of the uses of brackets is to order execution priorities in an R expression. Again, R treats them as a function.

```
> 30*(10+20)
[1] 900
> 30*"(10+20) # This is what is executed behind the scenes.
[1] 900
> expression(30*10+20)
expression(30 * (10 + 20))
> expression(30*"(10+20))
expression(30 * (10 + 20))
```

The same is true for the notion of expression blocks. An expression block is defined as a sequence of R expressions, grouped between curly bracket delimiters "{" and "}".

```
> {
+ print("line1")
+ print("line2")
+ }
[1] "line1"
[1] "line2"
> {"(print("line1"),print("line2"))
[1] "line1"
[1] "line2"
> expression({
+   print ("      "line1"      ) # This comment is not interpreted.
+
+   # Neither is this comment.
+   print("line2")
+ })
expression({
  print("line1")
  print("line2")
})
> expression( {"(print("line1"),print("line2"))  )
expression({
  print("line1")
  print("line2")
})
```

Note that comments and spaces are ignored by the R interpreter. Note also that to make your code easier to read, you can add as many carriage returns as you wish in a block without any effect on its execution.

8.4.2.2 R Formulae

The formula object is one of the specificities of R. It is mainly used to establish a relationship between two parts, separated with a tilde \sim . Both parts must be R expressions. Keeping in mind what we have learnt about the function `expression()`, we can see how R converts a formula into a " \sim "() function upon execution.

```
> y~x
y ~ x
> "~"(y,x)                      # Equivalent expression,
y ~ x
> expression("~"(y,x))    # as this expression proves.
expression(y ~ x)
```

For developers, formula objects can be used to offer a more user-friendly interface, since they are closer to the human language. For example, the R formula `y~x` can express that `y` and `x` are linked or that `y` is a function of `x`. Generally speaking, the developer bears the responsibility of interpreting the formula to perform the

necessary tasks. This is very advanced; we refer the interested reader to the R documentation files. Here are a few examples with no particular meaning, but which will help become familiar with this new object:

```
> y~x
y ~ x
> y~(x+y:z)*t|v
y ~ (x + y:z) * t / v
> y1+y2|w ~ (x+y:z)*t|v
y1 + y2 / w ~ (x + y:z) *t / v
```

It is worth pointing out that even if the quantities mentioned in the formulae above are not existing R objects, no error is thrown. However, remember that a syntax error results in an error message:

```
> y~x+y)*t|v
Error : ')' not expected in "y~x+y)"
```

We now focus on usage of formulae in the R system. Since formulae are not common objects, the user may not realize that they are saved like any other R object.

```
> form <- y~x
> form
y ~ x
```

The two main uses are for plots and for statistics.

For plots, this is an alternative to what we introduced in Chap. 7.

```
> x <- runif(10)
> y <- runif(10)
> plot(x,y)
> plot(y~x)
```

The resulting plot is not shown here, since the only interest is in showing that the instructions with or without the formula are equivalent. Note that the variables x and y are inverted between the two forms. The version with the formula `plot(y~x)` expresses more literally the action we want: plot y as a function of x. This version, which we find elegant, is of course also available for the complementary functions `points()` and `lines()`.

In a statistical context, a function relative to the specific treatment of a statistical model takes as input argument a formula establishing the relationship between the variables of the model (the formula is often the first argument). The most simple example is the linear model; here is an example¹¹:

```
> lm(y~x)    # x and y must be defined (and they are in this
               # case!)
Call:
lm(formula = y ~ x)
Coefficients:
(Intercept)          x
 0.46290        -0.06904
```

¹¹ This section does not give details on handling linear models in R; this will be the focus of Chap. 14.

```
> lm(form) # Recall that: form <- y~x
Call:
lm(formula = form)
Coefficients:
(Intercept)          x
0.46290      -0.06904
```

Besides the pleasant syntax, the formula object also offers a very efficient interface with the user to describe the model. This is confirmed by the fact that, unlike for plots, there is no other way of describing the relationship between the variables in the model. You might think that the syntax `lm(y, x)` could have been used. But then how would you write as a list of input arguments the formula `y~(x+z)*t`, which is perfectly valid (see Chap. 15)?

For operations on formulae, you can use the function `update()` which modifies a formula, using another one.

```
> update(y~x,.~.+z)      # Change y~x into y~x+z.
y ~ x + z
> form <- y~x           # The same procedure with a saved model.
> form2 <- update(form,.~.+z)
> form2
y ~ x + z
> update(form2,.~.-x)   # You can also delete a variable.
y ~ z
```

These examples show the syntax of the function `update()`. The first formal argument is the formula you wish to modify; the second formal argument gives the operations to perform on the formula, using a specific syntax. All that remains to be done is to analyse the syntax of the second formula. Any dot “.” before the tilde character “~” is replaced with the left expression of the initial formula (before the tilde). Similarly, any dot “.” after the tilde is replaced with the right expression of the initial formula (after the tilde).

8.4.2.3 The R Environment

The notion of environment is necessary in any programming language. It can be seen as a storage space of R objects. When you open your R session, a first environment `.GlobalEnv` is created by R. It is called the workspace and all objects manipulated with the command line during this session are stored there. Although we only wish to give an overview of this concept, it is worth mentioning that the notion of function depends intrinsically on the notion of environment. Here is a glimpse of this fact. When you create a new object in the body of a function, R takes care of declaring it internally in an environment specific to this function, to store the contents of the object. The reason for this is that if the object has the same name as an object of the environment `.GlobalEnv`, this last object will not be overwritten with the value defined in the body of the function. To better understand what an environment is, note that the value of an object defined in the environment `.GlobalEnv` can be

accessed in the body of the function. However, its value cannot be modified by an affectation with the same object name. The reason why you can access an object which was defined in another environment than the one associated with the function is that a parent environment is specified when declaring a new environment. It is allowed that an environment has no parent, as is the case with the initial environment `.GlobalEnv`. When an object is not directly available in the environment of a function, R searches for the object in the parent environment. If it is still not available, there are two possibilities: either there exists a “grandparent” environment, and the search continues, or there is no such environment and an error is thrown indicating that the object could not be found. This exploration process is repeated recursively until the object is found. Most environment declarations are done internally and invisibly by R. We shall return to this notion when we give more details on developing functions. A very surprising feature is that an environment is considered as an R object. A new environment can thus be declared to execute a specific block of code without changing the workspace `.GlobalEnv`. The function `local()`, which takes as first argument the code to execute and as second argument the environment for the execution, is very useful to this end:

```
> a <- 12; b <- 13
> space <- new.env() # By default, the parent is the environment
# from which new.env is called.
> local({
+ a <- b+2
+ a
+ },space)
[1] 15
> a # The value of a has not changed in .GlobalEnv.
[1] 12
> space$a # Value of a in the environment space.
[1] 15
```

The function’s name is well chosen: the value of `a` in the workspace `.GlobalEnv` has been preserved. As stated in the comment, the parent of `space` (generated by `new.env()`) is `.GlobalEnv`, but the parent could have been specified by giving a value to the formal argument `parent`. Here are two examples of parent declaration:

```
> space2 <- new.env(parent=emptyenv())
> local(a<-b+2,space2) # Error!!!
Error in eval(expr, envir, enclos) : could not find function "<-
> space2$a # Unsurprisingly, the object a does not exist!
NULL
```

The environment `space2` is useless, since its parent environment is an empty environment (i.e. no parent; declared with the function `emptyenv()`). The execution error in the `local` code is because even the affectation function `<-` cannot be accessed: the empty environment knows absolutely nothing about R; in particular, it does not know the basic functions. The function `globalenv()` returns the global environment `.GlobalEnv` which is always first in the access list of R environments.

```
> space3 <- new.env(parent=parent.env(globalenv()))
> local(a<-b+2,space3) # Error, because .GlobalEnv cannot be
# accessed!
Error in eval(expr, envir, enclos) : object 'b' not found
> local(a<-15,space3)
> a
[1] 12
> space3$a
[1] 15
```

Environments are rather convenient-they are used like a list.

```
> space3$b <- b-1
> b
[1] 13
> space3$b
[1] 12
```

For further details, we refer the reader to the online help, which is rather complete, but aimed at advanced users.

SECTION 8.5

† Interfacing R and C/C++ or Fortran

You may be wondering why you should consider writing parts of your code in C/C++ or Fortran. There are several reasons, such as:

- To use from R a pre-existing routine, formerly coded in C/C++ or Fortran
- To speed up the runtime of your R code
- To use the graphical capabilities of R or some R functions on numerical output from C/C++ or Fortran code

Tip

The last version of R includes a byte compiler which speeds up some computations. You can also use the R version distributed by the company **Revolution Analytics** (<http://www.revolutionanalytics.com>). It has been optimized to speed up some computations, for example, by relying on a multi-core architecture when available.



Warning

Interfacing R and C/C++ or Fortran is much more convenient under Linux (or MacOS) than under a Microsoft Windows OS for which several necessary tools lack. Note that the authors of this book use Linux on a daily basis!



See also

We assume that the reader already has some notions of C/C++ and/or Fortran programming. If that is not the case, the books [22, 38] for C and C++, and [9] for Fortran may be of use.



In this section, we do not claim exhaustivity. We shall only present a few simple examples which illustrate the points made above. Along the way, we shall provide some basics which we hope will allow you to get by on your own afterwards.

Warning

Before you start, you need to install C/C++ and Fortran compilers, since Microsoft Windows does not have any by default. The free software **Rtools**, containing several tools from the Linux world, has been created to this end. You can download it from <http://cran.r-project.org/bin/windows/Rtools>. Choose Full installation to build 32 or 64 bit R 2.14.2+ if you have a 64 bit processor. Tick the appropriate box when installing **Rtools**, so that the variable PATH is correctly configured. You also need to change the system environment variable Path so that it contains the path to the R installation folder (one way to find the path is to right-click on the R icon of the desktop, then choose properties). This will allow you to call R from an MS-DOS command window, as we shall mention later on. To do this, right-click on the Windows Desktop, select New/Shortcut, then enter the following instruction in the window that opens: `control.exe sysdm.cpl,System,3`



Once this shortcut has been created on the desktop, double-click on it, and in the window that opens, click on Environment Variables... Change the value on the system variable Path to add at the beginning (using ; as separator) the path to the folder containing the R executable (which should look like `C:\Program Files\R\R-3.1.0\bin\i386` or `C:\Program Files\R\R-3.1.0\bin\x64`) and the path to the folders of **Rtools** (which should look like `C:\Rtools\bin` and `C:\Rtools\gcc-4.6.3\bin`), if they are not already present.

8.5.1 Creating and Running a C/C++ or Fortran Function

The next example shows how to speed up a program by using C/C++ or Fortran. The R function `combn()` is able to handle all combinations of a given number of elements taken from a given vector. For example, this instruction generates all combinations of size 3 from the vector `1:5`.

```
> combn(5,3)
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2    3
[2,]    2    2    2    3    3    4    3    3    4    4
[3,]    3    4    5    4    5    5    4    5    5    5
```

If we attempt to get all `choose(n,m)` combinations (e.g., 1,313,400 combinations if $n = 200$ and $m = 3$) from a vector of larger size n , the computation time can increase drastically.

```
> system.time(x <- combn(200,3))
  user  system elapsed
14.959   0.227 15.188
```

The command `system.time()` shows that the above computation takes several seconds on the computer used to write this book (if your computer is faster, take a value greater than 200).

Tip

 The function `permn()` of package `combinat` can be used to generate all permutations of the elements of a vector.

A simplified version of the original R function `combn()` is given below:

```
> combnR <- function(n,m) {
+   a <- 1:m ; e <- 0 ; h <- m
+   combmat <- matrix(0,nrow=m,ncol=choose(n,m))
+   combmat[,1] <- 1:m
+   i <- 2
+   nmmp1 <- n - m + 1
+   mp1 <- m + 1
+   while (a[1] != nmmp1) {
+     if (e < n-h) {
+       h <- 1 ; e <- a[m] ; a[m-h+1] <- e + 1
+       combmat[,i] <- a
+       i <- i + 1
+     } else {
+       h <- h + 1 ; e <- a[mp1-h]
+       a[(m-h+1):m] <- e + 1:h
+       combmat[,i] <- a
+       i <- i + 1
+     }
+   }
+   return(compmat)
+ }
```

We now propose two functions coded in C/C++, and another two coded in Fortran, to make the same computation in much shorter time.

- Creating the C/C++ function

C++ code for function `combnC`, downloadable from <http://biostatisticien.eu/springeR/combn.cpp>:

```

1 #include <math.h>
2 extern "C" {
3 void combnC( int *compmat, int *n, int *m) {
4   int i, j, e, h, nmmp1, mp1;
5   int *a;
6   a=new int[m[0]];
7   for ( i=1;i<=m[0];i=i+1) a[i-1]=i ;
8   e=0;
9   h=m[0];
10  for ( i=1;i<=(m+0);i=i+1) combmat[i-1]=i ;
11  i=2;
12  nmmp1=n[0] - m[0] + 1;
13  mp1=m[0] + 1;
14  while(a[0] != nmmp1) {
15    if(e < n[0] - h) {
16      h=1;
17      e=a[m[0]-1];
18      a[m[0] - h]=e + 1;
19      for ( j=1;j<=m[0];j=j+1) combmat[(i-1)*m[0]+j-1]=a[j-1];
20      i=i+1;
21    } else {
22      h=h + 1;
23      e=a[mp1 - h-1];
24      for ( j=1;j<=h;j=j+1) a[m[0] - h + j-1]=e + j ;
25      for ( j=1;j<=m[0];j=j+1) combmat[(i-1)*m[0]+j-1]=a[j-1];
26      i=i + 1; }
27    delete [] a;
28  }}}

```

Code for the main function, downloadable from <http://biostatisticien.eu/springeR/main.cpp>:

```

1 #include <iostream>
2 using namespace std;
3 extern "C" {
4 int main() {
5   void combnC( int *compmat, int *n, int *m);
6   int *n, *m, *compmat, j;
7   double Cnm;
8   n=new int[1];
9   m=new int[1];
10  n[0]=5;
11  m[0]=3;
12  Cnm=10;
13  combmat=new int[(int)Cnm*m[0]];
14  combnC(compmat,n,m);
15  for ( j=1;j<=Cnm*m[0];j++) cout << combmat[j-1] << " ";
16  }}}

```

Note that all indices start at zero in C/C++, unlike R where they start at 1.

- Creating the Fortran function

Fortran code for the subroutine `combnF`, downloadable from <http://biostatisticien.eu/springeR/combn.f90>:

```

1 SUBROUTINE combnF(compmat ,n,m)
2
3 integer , intent(in) :: n,m
4 integer :: i,j,e,h,nmmp1,mp1
5 integer ,dimension(m) :: a
6 integer ,dimension(*), intent(out):: combmat
7
8 do i=1,m
9 a(i)=i
10 end do
11 e=0
12 h=m
13 do i=1,m
14 combmat(i)=i
15 end do
16 i=2
17 nmmp1=n-m+1
18 mp1=m+1
19 do while (a(1) .ne. nmmp1)
20 if (e < n-h) then
21 h=1
22 e=a(m)
23 a(m-h+1)=e+1
24 do j=1,m
25 combmat((i-1)*m+j)=a(j)
26 end do
27 i=i+1
28 else
29 h=h+1
30 e=a(mp1-h)
31 do 40 j=1,h
32 a(m-h+j)=e+j
33 40 continue
34 do j=1,m
35 combmat((i-1)*m+j)=a(j)
36 end do
37 i=i+1
38 endif
39 enddo
40 END SUBROUTINE combnF

```

Code for the main function, downloadable from <http://biostatisticien.eu/springeR/main.f90>:

```

1 PROGRAM main
2 integer :: n,m,Cnm,j ,k
3 integer , allocatable ,dimension (:) :: combmat
4 n=5
5 m=3
6 Cnm=10
7 k=Cnm*m
8 allocate (compmat(k))
9 CALL combnF(compmat ,n ,m)
10 write (* ,*) (compmat(j) ,j=1,k)
11 deallocate (compmat)
12 end PROGRAM main

```

- Compiling and running the C/C++ or Fortran function

In order to use the C++ or Fortran code given above, it needs to be compiled, i.e. transformed into an executable file. To do this, simply open an MS-DOS terminal window, for example, from the Windows menu Start/Run (or with the keyboard combination [WINDOWS+R]) and type the instruction cmd followed by ENTER. In this black window, type the two instructions below.

Warning

You may need to move to the directory where your files were saved, using the MS-DOS command cd (for *change directory*). For example, if you created your files on the Windows Desktop, use

`cd Desktop`

Note that under MS-DOS, the command dir is used to list the contents of the current directory.



```

:: To compile C/C++ code:
g++ -o mycombn.exe combn.cpp main.cpp
:: To compile Fortran code:
gfortran -o mycombn.exe combn.f90 main.f90
:: To run the function:
mycombn.exe

```

The first instruction compiles our C++ or Fortran code to produce the executable file `mycombn.exe`. The second instruction launches this executable file and displays, though with no formatting, the result of the computation.

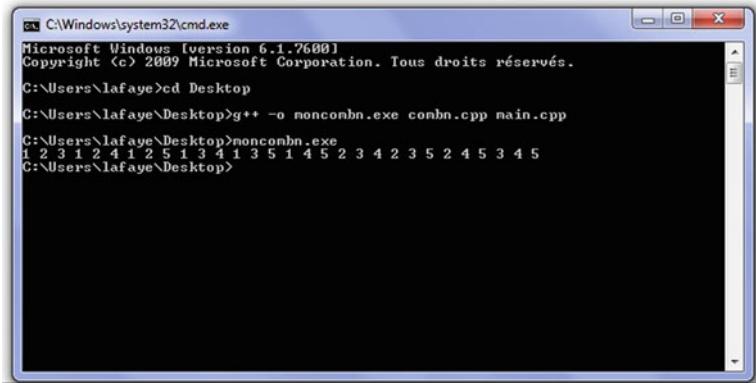
Tip

The function `system()` is used to execute a DOS command outside of R. For example, in R, type:



```
> system("mycombn.exe")
1 2 3 1 2 4 1 2 5 1 3 4 1 3 5 1 4 5 2 3 4 2 3 5 2 4 5 3 4 5 >
```

Note that you must of course first change the current R directory, using function `setwd()`, for example, to change to the directory containing the file `mycombn.exe`.

**Tip**


The compilation flag `-Wall` is used to display all compilation warnings or errors (if there are any!):

```
g++ -o mycombn.exe combn.cpp main.cpp -Wall
```

We shall now produce the $\binom{200}{3} = 1,313,400$ sub-vectors made of all possible combinations of three elements in vector `1:200`. For the C/C++ version, modify lines 11, 13 and 16 of the code of function `main` given p. 233. These lines become

```
n[0]=200;
Cnm=1313400;
// for (j=1;j<=Cnm[n0];j++) cout << combmat[j-1] << " ";
```

For the Fortran version, modify lines 4, 6 and 10 of the code of function `main` given p. 235. These lines become

```
n=200
Cnm=1313400
!write(*,*) (combmat(j) ,j=1,k)
```

We commented out the last line (using `//` in C/C++ and `!` in Fortran) so that a call of `mycombn.exe` no longer displays the (now very large) result of the computation, which would take a lot of time. But the calculation is made. We are thus coherent with the previous computation done in R, for which the result was not displayed but stored in variable `x`. After saving your changes, recompile and run your code:

```
:: To compile C/C++ code:  
g++ -o mycombn.exe combn.cpp main.cpp  
:: To compile Fortran code:  
gfortran -o mycombn.exe combn.f90 main.f90  
:: Execute the function:  
mycombn.exe
```

You can see that the calculation (without displaying the result) is done very quickly.

8.5.2 Calling C/C++ (or Fortran) from R

We shall now see how to call the C++ code from file `combn.cpp` (or rather a compiled version of this code) directly from R, without using a `main` function. To this end, we create an R wrapper containing a call of the C++ function.

Note

R can only call C/C++ or Fortran functions which do not return any output. All C/C++ functions must thus be of type `void` and all Fortran routines must be subroutines. The results will go in the arguments of the calling function.



Download the file <http://biostatisticien.eu/springeR/combn.R>, which includes the code given below:

```
1 combnRC <- function(n,m) {  
2   combmat <- matrix(0,nrow=m,ncol=choose(n,m))  
3   lib <- paste("combn",.Platform$dynlib.ext,sep=".")  
4   dyn.load(lib)  
5   out <- .C("combnC",res=as.integer(compmat),  
6             as.integer(n),as.integer(m))  
7   combmat <- matrix(out$res,nrow=m,byrow=F)  
8   dyn.unload(lib)  
9   return(compmat)  
10 }
```

Note

To call the Fortran code, replace line 5 by

```
out <- .Fortran("combnF", res=as.integer(compmat),
```



The functions `dyn.load()` and `dyn.unload()` allow respectively to load and unload from R's memory the resources from a DLL (dynamic link library) file. A DLL includes functions which can be called during the execution of a program, without being included in its executable. Here, it is the file `combn.dll` (which includes only one function), which will be created further on.

The functions `.C()` and `.Fortran()` (which output a list) are used to send values from R to a C/C++ or Fortran function, respectively. Use the instructions `as.integer()`, `as.double()` or `as.character()` in R to declare objects made of integer values, decimal (numeric) values or character strings, so that they are “received” correctly by the arguments of the C/C++ or Fortran function.

For a C/C++ function, all arguments must be pointers, for example, integer pointers (`int *`), real pointers (`double *`) or character pointer pointers (`char **`). Table 8.1 gives the equivalent types in R, C/C++ and Fortran.

Table 8.1: Conventions on argument types. Type `? .Fortran` for further detail

R	C/C++	Fortran
<code>integer</code>	<code>int *</code>	<code>INTEGER</code>
<code>numeric</code>	<code>double *</code>	<code>DOUBLE PRECISION</code>
<code>numeric</code>	<code>float *</code>	<code>REAL</code>
<code>complex</code>	<code>Rcomplex *</code>	<code>DOUBLE COMPLEX</code>
<code>logical</code>	<code>int *</code>	<code>integer</code>
<code>character</code>	<code>char **</code>	<code>CHARACTER*255</code>
<code>list</code>	<code>SEXP *</code>	not allowed
<code>other type</code>	<code>SEXP</code>	not allowed

Warning

Unlike R, where it is very easy to get the length of vector `x` with the instruction `length(x)`, in C/C++ it is not possible to know the length of `x`. It can sometimes be useful to give to the function `.C()` both the vector `x` and its length, for example, as follows for some hypothetical function `functionC`:

```
x <- c(1.2, 0.7, 3, 2, 4, 1, 0.9)
.C("functionC", as.double(x), as.integer(length(x)))
```



The arguments of the C/C++ function `functionC` are `double *x` and `int *n`. The same remark applies to Fortran functions.

Note

The C/C++ function `combnC` returns `void`: it does not have any direct output. However, the value of its arguments, which are pointers, can be modified during execution. It is then possible to access directly (thanks to their address) to the value of these pointers. This is how R works, using the function `.C()` (in a transparent way for the user).

You may have noted at line 5 of the code of function `combnRC()` above that we used `res=` when calling function `.C()`. This allows us to use `out$res` directly, instead of `out[[1]]`. You can use another name than `res`, and for any argument of function `.C()`. For example, we could have used `val=as.integer(m)`, which we did not do because that value was not modified by `combnC` and is thus already known (as `m`). A similar remark applies to **Fortran** functions.



We shall now create the file `combn.dll`, which will be called by R. To this end, type the following instructions in an MS-DOS window:

```
:: In C/C++:  
g++ -c combn.cpp -o combn.o  
g++ -shared -o combn.dll combn.o  
:: In Fortran:  
gfortran -c combn.f90 -o combn.o  
g++ -shared -o combn.dll combn.o
```

Tip

Equivalently (or almost equivalently, since optimization arguments could be used by the compiler, which might by the way hinder debugging), this dynamical library could be created (after deleting if necessary the files `combn.o` and `combn.dll`) with one instruction:



```
R CMD SHLIB combn.cpp -o combn.dll
```

The first instruction creates the object file `combn.o`, which contains the machine code for the function included in file `combn.cpp`. The second instruction creates the dynamic library `combn.dll`. At this step, the compiler informs us of any errors to correct in the program (with the corresponding line number).

Tip

Note that it is possible to include several object files in the same library, which will then contain several functions. For example, if we had a file `choose.o` containing the machine code for a function which calculates binomial coefficients, we could include both functions in a DLL as follows:

```
g++ -shared -o combn.dll combn.o choose.o
```

**Linux**

Under Linux, DLL files usually have a `.so` extension (for *shared object*). You should thus replace all occurrences of extension `.dll` by extension `.so`.

Mac

Under Mac OS, DLL files usually have a `.dylib` extension (for *dynamic library*). You should thus replace all occurrences of extension `.dll` by extension `.dylib`. Also note that under Mac OS, you must replace `g++ -shared` with `g++ -dynamic`.

In R, after changing to the correct directory, we can now execute the following instructions:

```
> combn(5,3)
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2    3
[2,]    2    2    2    3    3    4    3    3    4    4
[3,]    3    4    5    4    5    5    4    5    5    5
> source("combn.R")
> combnRC(5,3)
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2    3
[2,]    2    2    2    3    3    4    3    3    4    4
[3,]    3    4    5    4    5    5    4    5    5    5
> system.time(x <- combn(200,3))
  user  system elapsed
14.803   0.229 15.035
> system.time(x <- combnRC(200,3))
  user  system elapsed
0.158   0.023  0.181
```

There is an important speed-up, thanks to this new R function using code written in C/C++.

Do it yourself

Code in **R** alone, and then in hybrid **R-C/C++** (or **R-Fortran**), the functions `ar1simR()` and `ar1simRC() - ar1simC` (or `ar1simRF() - ar1simF`). These functions take three input arguments: $n \in \mathbb{N}$, $\phi \in (-1, 1)$ and $M \in \mathbb{N}$. They do the following computations.

For $m = 1, \dots, M$:

- Simulate random vector $\epsilon = (\epsilon_1, \dots, \epsilon_n)^\top$ with distribution $\mathcal{N}_n(\mathbf{0}; \mathcal{I}_n)$.
- Create vector $x = (x_1, \dots, x_n)^\top$, with $x_1 = \epsilon_1$, and such that for all $t = 2, \dots, n$, we have $x_t = \phi x_{t-1} + \epsilon_t$.
- Calculate the conditional least squares estimator $\hat{\phi}_m$ of ϕ :

$$\hat{\phi}_m = \frac{\sum_{t=2}^n x_{t-1} x_t}{\sum_{t=2}^n x_{t-1}^2}.$$

The functions you create should output the value $\bar{\hat{\phi}} = \frac{1}{M} \sum_{m=1}^M \hat{\phi}_m - \phi$, thus allowing a numerical evaluation of the bias of estimator $\hat{\phi}$ of ϕ .

Compare the speed of execution of the pure **R** version with the version calling **C/C++** (or **Fortran**) code. To this end, plot the values $(M, time_M)$ for $M = 1,000, 2,000, \dots, 100,000$. Take $n = 1,000$ and $\phi = 0.75$.

Note: The function `arima.sim()` performs parts (a) and (b) above, and function `arima()` performs part (c). Do not use these two pre-existing functions for this exercise: they are very fast because they are coded in **C**, but are not limited to the previous computations.

Tip

To ease code development, a good editor is always useful. An editor should at least include indentation and syntactical colouring. You may wish to use the following free software:

- An **R** code editor such as **RStudio**, **Tinn-R** or **Emacs**
- A source code editor for **C/C++** and **Fortran** such as **Emacs** or **Code::Blocks** (available at <http://www.codeblocks.org>)



Tip

The package **rbenchmark** can be used to easily calculate the expected gain in computation time by using an **R-C/C++** or **R-Fortran** function rather than a pure **R** function. For example, try to verify the results we got in the previous practical using the following code:



```
n <- 1000
phi <- 0.75
M <- 2000
dyn.load("ar1sim.dll")
benchmark(Rcode=ar1simR(n,phi,M),
          Ccode=.C("ar1simC",as.integer(n),phi,
                  as.integer(M),res=0.0)$res,
          replications=1000)
```

Tip


Fortran and **R** store matrices (tables) in the same way: the rows of a given column are stored sequentially in memory. In **C/C++**, the opposite holds; columns of a given line are stored sequentially. Be careful when sending a matrix from **R** to **C/C++**. For example, the element with index $[i, j]$ in an **R** matrix corresponds to the element with index $[(j-1)*\text{number-of-rows} + (i-1)]$ in **C/C++** (in **C/C++**, indices start at 0).

8.5.3 Calling External C/C++ or Fortran Libraries

It is possible to use a function from an **external** library, thanks to the **R** functions **.C()** (for **C/C++** libraries) and **.Fortran()** (for **Fortran** libraries).

Tip


Here is an amusing application of this approach, which locks the Windows session:



```
# Select file C:/windows/system32/user32.dll:
dyn.load(file.choose())
.C("LockWorkStation")
```

It is also possible to call an external library directly from your **C/C++** or **Fortran** code. Here are some scientific libraries which we find interesting:

- The R API (*application programming interface*)
- The C++ library `newmat`
- The Fortran libraries BLAS and LAPACK

See also

Other libraries exist; some are free of charge, or even open-source, such as:

- In C/C++:
 - <http://www.gnu.org/software/gsl>
 - <http://www.math.uiowa.edu/~dstewart/meschach>
 - <http://www.nrbook.com/a/bookcpdf.php>
- In Fortran:
 - <http://calgo.acm.org>
 - <http://www.nrbook.com/a/bookfpdf.php>
 - <http://www.nrbook.com/a/bookf90pdf.php>
 - <http://math-atlas.sourceforge.net>

Others are not free:

- In C/C++:
 - <http://www.nag.co.uk/numeric/CL/CLdescription.asp>
 - <http://www.vni.com/products/imsl/c/imslc.php>
- In Fortran:
 - <http://www.nag.co.uk/numeric/RunderWindows.asp>
 - <http://www.nag.co.uk/numeric/fl/FLdescription.asp>
 - <http://www.nag.co.uk/numeric/fn/FNdescription.asp>
 - <http://www.vni.com/products/imsl/fortran/overview.php>

8.5.3.1 The R API

The R API is a library created by the R developers. It can be used from a C/C++ program without even using R (it is then called standalone R API). It can also be used in C/C++ code which will itself be called from R, as introduced in the previous section. This allows the use of existing routines without having to rewrite them. To use this library, you must include in your C/C++ source code the two header files `R.h` and `Rmath.h`, which are necessary to declare or define some mathematical functions and constants.

See also

The documentation for this library, which includes the list of functions and constants contained in it, is available at <http://cran.r-project.org/doc/manuals/R-exts.html#The-R-API>.

You may also find interesting to consult the contents of the directory `nmath` in the R sources; it is available at <http://svn.r-project.org/R/trunk/src/nmath>.

We present below C/C++ code available at <http://biostatisticien.eu/springeR/integ.cpp> which allows to compute the integral

$$\int_{\epsilon_1}^{\pi} \Phi(t + \epsilon_2) dt,$$

where ϵ_1 and ϵ_2 are realizations of two independent random variables (respectively, normal and uniform) and where $\Phi(\cdot)$ is the cumulative distribution function of the $\mathcal{N}(0, 1)$ distribution. The only point of this example is to illustrate the use of the R API to simulate random variables, calculate a probability and perform numerical integration.

```

1 #include <R.h>
2 #include <Rmath.h>
3
4 extern "C" {
5
6     typedef void integr_fn(double *x, int n, void *ex);
7     void f(double *t, int n, void *ex);
8     void testintegral(double *res) {
9
10         // R API numerical integration function
11         void Rdqags(integr_fn f, void *ex, double *a,
12                     double *b, double *epsabs,
13                     double *epsrel, double *result,
14                     double *abserr, int *neval,
15                     int *ier, int *limit, int *lenw,
16                     int *last, int *iwork, double *work);
17
18         GetRNGstate(); // Read the R generator seed
19
20         double *a, *b, *epsabs, *epsrel, *result,
21             *ex, *abserr, *work;
22         int *last, *limit, *lenw, *ier, *neval, *iwork;
23
24         ex = new double[1]; a = new double[1];
25         b = new double[1]; epsabs = new double[1];

```

```
26     epsrel = new double[1]; result = new double[1];
27     abserr = new double[1]; neval = new int[1];
28     ier = new int[1]; limit = new int[1];
29     lenw = new int[1]; last = new int[1];
30     limit[0] = 100;
31     lenw[0] = 4 * limit[0];
32     iwork = new int[limit[0]];
33     work = new double[lenw[0]];
34
35     a[0] = rnorm(0.0, 1.0); // eps1 from N(0,1)
36     b[0] = M_PI; // The constant \pi (3.141593...)
37     ex[0] = runif(0.0, 1.0); // eps2 from Unif(0,1)
38
39 // Calculate the integral
40 Rdqags(f, ex, a, b, epsabs, epsrel,
41         result, abserr, neval, ier,
42         limit, lenw, last,
43         iwork, work);
44
45 // The result is stored in res[0]
46 res[0] = result[0];
47
48 PutRNGstate(); // Write the generator seed
49
50 // Free up some memory
51 delete[] ex, a, b, epsabs, epsrel, result, abserr,
52 neval, ier, limit, lenw, last, iwork, work;
53 }
54
55 // Define the function to integrate
56 void f(double *t, int n, void *ex) {
57     int i;
58     double eps2;
59     eps2 = ((double*)ex)[0];
60     for (i=0;i<n;i++) {
61         t[i] = pnorm(t[i]+eps2, 0.0, 1.0, 1, 0);}
62 }
63
64 }
```

The instructions to compile this function in order to get a DLL file are

```
g++ -c integ.cpp -o integ.o -I"C:\Program Files\R\R-3.1.0
\include"
g++ -shared -o integ.dll integ.o ^
-L"C:\Program Files\R\R-3.1.0\bin\i386" -lR
```

Warning

Note that we had to indicate the paths to the folders containing the files R.h, Rmath.h and R.dll. Modify these as needed depending on your system configuration. In MS-DOS, the symbol ^ indicates an incomplete line.

Linux

```
g++ -c integ.cpp -o integ.o -I"/usr/lib/R/include" -fPIC
g++ -shared -o integ.so integ.o -I"/usr/lib/R/include" \
-L"/usr/lib" -lR
```

Now, to perform the calculation in R, use the following instructions:

```
> dyn.load(paste("integ", .Platform$dynlib.ext, sep=""))
> # i.e. dyn.load("integ.dll") under Windows.
> .C("testintegral", val=0.0)$val
[1] 3.707762
```

Of course, the result of this computation varies, depending on the realizations of ϵ_1 and ϵ_2 .

8.5.3.2 The newmat Library

The newmat library is used to manipulate various types of matrices and to perform classical operations such as multiplication, transposition, inversion, eigenvalue computation and decompositions.

See also

The complete documentation for this library is available at <http://www.robertnz.net/nm11.htm>.

The code below, available at <http://biostatisticien.eu/springeR/inv.cpp>, is C/C++ code using this library to invert a matrix and can be called from R.

```
1 #define WANT_STREAM
2 #define WANT_MATH
3 #include "newmatap.h"
4 #include "newmatio.h"
5 #ifdef use_namespace
6 using namespace NEWMAT;
7 #endif
8 extern "C" {
9     void invC(double * values, int * nrow) {
```

```

10 int i , j ;
11 Matrix M(nrow[0],nrow[0]);
12 M << values ;
13 M << M.i () ; // Calcul de l'inverse de M
14 for ( i=1;i<=nrow[0]; i++) {
15     for(j=1;j<=nrow[0];j++) {
16         values [ nrow[0]*(i-1)+j -1] = M(i , j );
17     }
18 }
19 M.Release ();
20 return ;
21 }
22 }
```

Tip

Download file <http://www.robertnz.net/ftp/newmat11.zip> and unzip it in C:/newmat. Then type the following instructions in an MS-DOS window:

```

cd \
cd newmat
g++ -O2 -c *.cpp
ar cr newmat.a *.o
ranlib newmat.a
cp newmat.a newmat.dll
```



After a few minutes, the libraries newmat.a and newmat.dll are created in folder C:\newmat.

You now need to create the library inv.dll (or inv.so under Linux) using the following instructions:

```

cd folder containing file inv.cpp
g++ -IC:\newmat -o inv.o -c inv.cpp
R CMD SHLIB inv.cpp -IC:\newmat C:/newmat/newmat.a
```

Linux

```

g++ -I/usr/include/R -I/usr/local/include -Inewmat -fpic \
-c inv.cpp -o inv.o
R CMD SHLIB inv.cpp -Inewmat newmat/newmat.a
```



You can then use the C/C++ above from R as follows. First save the following code in a file called inv.R:

```
> inv <- function(M) {
+   n <- nrow(M)
+   return(matrix(.C("invC", Minv=as.vector(M), n) $Minv,
+   nrow=n, ncol=n))}
```

Then execute the instructions:

```
> dyn.load(paste("inv", .Platform$dynlib.ext, sep=""))
> A <- matrix(rnorm(9), nrow=3)
> solve(A) # The R function solve() inverts a matrix.
      [,1]      [,2]      [,3]
[1,] -0.09893572  0.04676191  1.155500
[2,] -0.47035376  1.10728717 -2.979609
[3,]  0.03415044 -1.07683806  1.456918
> inv(A)
      [,1]      [,2]      [,3]
[1,] -0.09893572  0.04676191  1.155500
[2,] -0.47035376  1.10728717 -2.979609
[3,]  0.03415044 -1.07683806  1.456918
```

The two functions `solve()` and `inv()` thus give the same result for matrix inversion. As you can see, the speed-up for this operation is substantial.

```
> benchmark(Rcode=solve(A), Ccode=inv(A), replications=10000)
  test replications elapsed relative user.self sys.self
2 Ccode        10000    0.255  1.000000     0.256     0.000
1 Rcode        10000    1.378  5.403922     1.351     0.025
  user.child sys.child
2          0          0
1          0          0
```

8.5.3.3 The BLAS and LAPACK Packages

The BLAS (*Basic Linear Algebra Subprograms*) and LAPACK (*Linear Algebra PACKAGE*) packages are Fortran packages which perform many matrix operations. We shall see how to use them on a simple example.

First download the archiver software 7-zip available at <http://www.7-zip.org/download.html>. Use this software (twice) to unzip (in two steps) the file <http://www.netlib.org/lapack/lapack.tgz>. All files and subfolders (BLAS, CMAKE, etc.) should be placed directly in a folder called `C:\lapack`. For example, this folder will contain at its root a file called `make.inc.example`, which you must rename to `make.inc` after changing the line `SHELL = /bin/sh` to `SHELL = sh`. Then type the following instructions in an MS-DOS window:

```
cd C:\lapack
make lapacklib blaslib
```

After several minutes, the static packages `librefblas.a` and `liblapack.a` are created.

See also

The documentation for these packages can be read at <http://www.netlib.org/lapack/lug>. It is also useful to read the source code of all BLAS and LAPACK routines you wish to use, as they contain a detailed description of the arguments the routines take.



Here is the Fortran code, also available at <http://biostatisticien.eu/springeR/inv.f90>, for a subroutine which computes the inverse of a matrix. It uses the subroutines `external DGETRF` and `DGETRI` from the Lapack package.

```
1 ! Returns the inverse of a matrix calculated by finding
2 ! the LU decomposition . Depends on LAPACK.
3 subroutine invF(A,Ainv,m)
4   double precision , dimension(m,m) , intent(in ) :: A
5   double precision , dimension(size(A,1),size(A,2)) , &
6                   intent(inout ) :: Ainv
7
8   ! work array for LAPACK
9   double precision , dimension(size(A,1)) :: work
10  integer , dimension(size(A,1)) :: ipiv ! pivot indices
11  integer :: n , info , m
12
13  ! External procedures defined in LAPACK
14  external DGETRF
15  external DGETRI
16
17  ! Store A in Ainv to prevent it from
18  ! being overwritten by LAPACK
19  Ainv = A
20  n = size(A,1)
21
22  ! DGETRF computes an LU factorization of
23  ! a general M-by-N matrix A using partial
24  ! pivoting with row interchanges.
25  call DGETRF(n , n , Ainv , n , ipiv , info)
26
27  if (info /= 0) then
28    stop 'Matrix is numerically singular!'
29  end if
30
31  ! DGETRI computes the inverse of a matrix using
32  ! the LU factorization computed by DGETRF.
33  call DGETRI(n , Ainv , n , ipiv , work , n , info)
34
```

```

35 | if (info /= 0) then
36 |   stop 'Matrix inversion failed !'
37 | end if
38|end subroutine invF

```

To compile this code, execute the following instructions from an MS-DOS window:

```

cd %HOMEPATH%/Desktop # To be changed to suit your needs.
gfortran -c inv.f90 -o inv.o -I"C:/lapack"
gfortran -shared -o inv.dll inv.o -I"C:/lapack" ^
          C:/lapack/liblapack.a C:/lapack/librefblas.a

```

Linux



Under Linux, use the following instructions:

```

gfortran -c inv.f90 -o inv.o -fPIC
gfortran -shared -o inv.so inv.o /usr/lib64/liblapack.so.3

```

After creating the file `inv.dll` (or `inv.so` under Linux) with the previous instructions, you can start R and type the following instructions:

```

> dyn.load(paste("inv", .Platform$dynlib.ext, sep=""))
> A <- matrix(rnorm(4), nrow=2)
> B <- matrix(0, nrow=2, ncol=2)
> .Fortran("invF", A, res=B, 2L)$res
      [,1]      [,2]
[1,] -1.1812737  1.9822527
[2,] -0.1681507 -0.7224351
> solve(A)
      [,1]      [,2]
[1,] -1.1812737  1.9822527
[2,] -0.1681507 -0.7224351

```

8.5.3.4 Mixing C/C++ and Fortran Packages

It is possible to call C/C++ functions from Fortran code, thanks to the instruction `F77_SUB(name)`. We illustrate this point in the next example, which generates two independent observations: one from a $\mathcal{N}(0, 1)$ distribution and the other from the uniform distribution. The Fortran code below uses the C functions `GetRNGstate`, `PutRNGstate`, `rnorm` and `runif` from the R API, which we have already used in Sect. 8.5.3.1. Save it in a file called `random.f`.

```

1 SUBROUTINE random(x,y)
2   real*8 normrnd, unifrnd, x, y
3   CALL rndstart()
4   x = normrnd()

```

```

5      y = unifrnd()
6      CALL rndend()
7      RETURN
8      END

```

Then create the file `random.c` containing

```

1 #include <R.h>
2 #include <Rmath.h>
3 void F77_SUB(rndstart)(void) { GetRNGstate(); }
4 void F77_SUB(rndend)(void) { PutRNGstate(); }
5 double F77_SUB(normrnd)(void) { return rnorm(0,1); }
6 double F77_SUB(unifrnd)(void) { return runif(0,1); }

```

To create your DLL file, compile using the instructions

```

gfortran -c random.f -o randomf.o
gcc -c random.c -o randomc.o -I"C:\Program Files\R\R-3.1.0\include"
\include" gfortran -shared randomf.o randomc.o -o random.dll ^
-L"C:\Program Files\R\R-3.1.0\bin\i386" -lR

```

Linux

Under Linux, use

```

gfortran -c random.f -o randomf.o -fPIC
gcc -c random.c -o randomc.o -I"/usr/lib/R/include" -fPIC
gfortran -shared randomf.o randomc.o -o random.so

```



You can now call your code from R using the instructions:

```

> dyn.load(paste("random", .Platform$dynlib.ext, sep=""))
> .Fortran("random", as.double(1), as.double(1))
[[1]]
[1] 1.542474
[[2]]
[1] 0.59143

```

It is also possible to call Fortran functions from C/C++ code, using the following instructions:

`F77_NAME(name)` to declare a Fortran routine in C

`F77_CALL(name)` to call a Fortran routine from C

`F77_COMDECL(name)` to declare a COMMON FORTRAN block in C

`F77_COM(name)` to access a COMMON FORTRAN block from C

Here is a small example (with Fortran77 for a change). Save the code below in a file called `combnCF.cpp`:

```

1 #include <R.h>
2 #include <Rmath.h>
3 extern "C" {
4 void combnCF(int *compmat, int *n, int *m) {
5 // Caution! No upper case in the name of the subroutine.
6 void F77_NAME(combnf)(int *compmat, int *n, int *m);
7 F77_CALL(combnf)(compmat,n,m);
8 }
9 }
```

Then type the following instructions in an MS-DOS command window to create the package which will be called from R:

```

g++ -c combnCF.cpp -o combnCF.o -I"C:\Program Files\R
\{R-3.1.0\include" gfortran -c combn.f90 -o combn.o
g++ -shared -o combnCF.dll combnCF.o combn.o ^
-L"C:\Program Files\R\R-3.1.0\bin\i386" -lR
```

Linux

Under Linux



```

g++ -c combnCF.cpp -o combnCF.o -I"/usr/lib/R/include"-fPIC
gfortran -c combn.f90 -o combn.o -fPIC
g++ -shared -o combnCF.so combnCF.o combn.o \
-I"/usr/lib/R/include" -L"/usr/lib" -lR
```

Now modify the code of function `combnRC()` given p. [237](#):

- Change the name of this function to `combnRCF()`.
- Replace "combn" and "combnC" with "combnCF".

Save this new code in a file called `combnCF.R`. Then type the following instructions in the R console:

```

> source("combnCF.R")
> combnRCF(5,3)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2    3
[2,]    2    2    2    3    3    4    3    3    4    4
[3,]    3    4    5    4    5    5    4    5    5    5
```

8.5.4 Calling R Code from a C/C++ Program Called by R

We have seen how to call a C/C++ (or Fortran) routine from R. It is also possible to use a type of pointer called **SEXP** (for *Simple EXPression*) and the function

.Call(). In this subsection, we only give a simple example. The reader can use this as inspiration for more complex examples.

See also

We refer the reader to the website <http://cran.r-project.org/doc/manuals/R-exts.html#Handling-R-objects-in-C>.



In the following example, we shall see how to call function pmvt() of package mvtnorm from C/C++ code, itself called from R. The function pmvt() computes the probability that a random vector following a multivariate Student distribution belongs to a specified hyperrectangle in \mathbb{R}^n .

Unlike the examples in the previous sections, which used the function .C(), we shall need the function .Call(). Furthermore, our C/C++ code will have to be a function (which we call pmvtC in the following) which returns a structure of type SEXP and which also takes arguments of type SEXP. The code below, available from <http://biostatisticien.eu/springeR/pmvtp.cpp>, will be transformed into a DLL file and then called by the function .Call().

```

1 #include <R.h>
2 #include <Rdefines.h>
3 #include "Rmath.h"
4 #include <R_ext/Rdynload.h>
5 extern "C" {
6     SEXP pmvtCR(SEXP Rupper,SEXP Rcorr,SEXP Rdf,
7                  SEXP Rpmvt,SEXP Renv,SEXP Rres) {
8         SEXP R_fcall;
9         if (!isFunction(Rpmvt) & (Rpmvt != R_NilValue))
10             error("Rpmvt must be a function");
11         if (!isEnvironment(Renv))
12             error("Renv must be an environment");
13         PROTECT(R_fcall = lang4(Rpmvt,Rupper,Rcorr,Rdf));
14         REAL(Rres)[0] = REAL(eval(R_fcall, Renv))[0];
15         UNPROTECT(1);
16         return (Rres);
17     }
18 }
```

To compile this file, use the following instructions:

```

g++ -c pmvt.cpp -o pmvt.o -I"C:\Program Files\R\R-3.1.0
\include"
g++ -shared -o pmvt.dll pmvt.o ^
-L"C:\Program Files\R\R-3.1.0\bin\i386" -lR
```

Linux

Under Linux, use the instructions



```
g++ -m64 -I/usr/include/R -I/usr/local/include -fpic \
     -c pmvt.cpp -o pmvt.o
R CMD SHLIB pmvt.cpp
# or:
g++ -m64 -shared -L/usr/local/lib64 -o pmvt.so pmvt.o \
     -L/usr/lib64/R/lib -lR
```

You can now call this function from R. First download the file

<http://biostatisticien.eu/springeR/pmvt.R> which contains the following code:

```
> pmvtRCR <- function(upper,corr,df) {
+   res <- 0.0
+   Rpmvt <- function(upper,corr,df) {
+     d <- length(upper)
+     pmvt(lower=rep(-Inf,d),upper=upper,delta=rep(0,d),
+       corr=matrix(corr,ncol=d),df=df)}
+   dyn.load(paste("pmvt",.Platform$dynlib.ext,sep=""))
+   res <- .Call("pmvtCR",as.double(upper), as.double(corr),
+                 as.double(df),Rpmvt,new.env(),as.double(res))
+   dyn.unload(paste("pmvt",.Platform$dynlib.ext,sep=""))
+   return(res)
+ }
```

Then type the following instructions:

```
> require("mvtnorm")
> corr <- diag(3)
> set.seed(1)
> source("pmvt.R")
> pmvtRCR(c(2,3,2),corr,c(1,1,1))
[1] 0.706062
> set.seed(1)
> pmvt(lower=rep(-Inf,3),upper=c(2,3,2),corr=corr,df=c(1,1,1)) [1]
[1] 0.706062
```

Tip



If an SEXP object contains a vector (e.g., SEXP x) or a matrix (e.g., SEXP M), you can use the instructions `R_len_t n = length(x)` and `R_len_t p = nrows(M)` to create integers containing the length n of vector x or the number of rows p of matrix M. The file `Rinternals.h` contains the list of many similar useful functions.

8.5.5 Calling R Code from Fortran

We recommend the open-source software RFortran available at <http://www.rfortran.org>.

8.5.6 Some Useful Functions

Here are a few functions which you may find useful. The following functions are used in an MS-DOS terminal window (or in Cygwin, see p. 258):

- `nm`: list of symbols of object files (e.g., `nm random.dll`).
- `objdump`: information about object files (e.g., `objdump -x random.dll`).
- `ldd`: list dynamic dependencies if necessary (e.g., `ldd random.dll`).

The following functions are used in R:

- `getLoadDLLs()`: list all DLLs loaded in the current session (e.g., `getLoadDLLs()`)
- `is.loaded()`: checks whether a library is loaded (e.g., `is.loaded(random.dll)`)

SECTION 8.6

† Debugging Functions

In this section, we present various options which can be useful to debug a function and find an error. The error could be either in the R code or in C/C++ or Fortran code called from your R function.

See also

We refer the reader to the website <http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR>.

8.6.1 Debugging Functions in Pure R

We present some debugging functions, useful when writing R code.

The Function `browser()`

A useful debugging function in R is the function `browser()`. If you insert the instruction `browser()` in the source of your function, the program will stop at the place where it was inserted.

Here is an example showing how to use `browser()` in a function called `lsq()` which calculates the least squares estimator of unknown arguments in a simple linear model (see Chap. 14 for further details).

```

1 lsq <- function(X,Y,intercept=TRUE){
2   X <- as.matrix(X)
3   Y <- as.matrix(Y)
4   plot(X,Y)
5   nbunits <- nrow(X)
6   browser()
7   if (intercept==TRUE) X <- cbind(rep(1,nbunits),X)
8   betahat <- solve(t(X) %*% X) %*% t(X) %*% Y
9   curve(betahat[1]+betahat[2]*x,add=TRUE)
10
11 return(betahat)
12 }
```

Source the file containing the previous code (e.g., with the instruction `source(file.choose())`), then type:

```
lsq(X=cars[,2],Y=cars[,1])
```

As you can see, the program stops and you can examine the contents of all local variables defined before `browser()`. For example, type `nbunits`.

Note



By typing the letter `n` (for *next*), you can inspect the code and the contents of variables sequentially. To leave the inspection mode, type `Q`.

Here is an overview of a debugging session:

```

lsq(X=cars[,2],Y=cars[,1])
Called from: mc(X = cars[, 2], Y = cars[, 1])
Browse[1]>nbunits
[1] 50
Browse[1]> betahat
Error: Object "betahat" not found
Browse[1]> n
debug: if (intercept == T) X <- cbind(rep(1, nbunits), X)
Browse[1]> n
debug: betahat <- solve(t(X) %*% X) %*% t(X) %*% Y
Browse[1]> n
debug: curve(betahat[1] + betahat[2] * x, add = T)
Browse[1]> betahat
[,1]
[1,] 8.2839056
[2,] 0.1655676
Browse[1]> Q
>
```

Note

If you enter the letter **c** (for *continue*), the code is executed until the end, unless a **browser()** command is met again.



The Function **debug()**

Another interesting function is **debug()** which is equivalent to putting the instruction **browser()** at the top of a function. Thus **debug(var)** marks the functions **var** as debuggable. Any subsequent call of this function will launch the online debugger.

```
debug(var)
var(1:3)
```

To get rid of this mark, use the function **undebug()**.

```
undebug(var)
```

8.6.2 Error in R Code

First change line 6 of file **combn.R**, replacing the affectation arrow **<-** by the symbol **<**. We now have an error: an omitted symbol (the symbol **-**):

```
compmat<matrix(out$res,nrow=m,byrow=F)
```

Save the file, source it and type the following instruction:

```
> combnRC(5,3)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]     0     0     0     0     0     0     0     0     0     0
[2,]     0     0     0     0     0     0     0     0     0     0
[3,]     0     0     0     0     0     0     0     0     0     0
```

As you can see, there is an error in the result, and the error that we introduced deliberately in the code could be difficult to detect if it were an accidental omission. Here is how we could try to detect where the error comes from. First install and load the package **debug**. Then use the function **mtrace()** of this package, as follows:

```
mtrace(combnRC)
combnRC(5,3)
```

You should then see a debugging window with the source code of function **combnRC()**. Pressing the RETURN key repeatedly will evaluate your source code line by line until the next display (which occurs upon evaluation of the line we modified):

```
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[2,] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[3,] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

This hints that there is an issue at this point. We can then correct the error, for example, with the instruction `fix(combnRC)`.

Note that the function `mtrace()` did not allow us to delve into the details of the following call:

```
.C("combnC",res=as.integer(compmat),as.integer(n),
  as.integer(m))
```

8.6.3 Error in the C/C++ or Fortran Code

We shall now see how to perform the same kind of debugging for parts of the code written in C/C++ or **Fortran**. It mostly boils down to using the compilation option `-g` to add information on the source code in the DLL file, and then to using a specialized debugging tool.

Warning

You will need a debugging tool. We recommend the free software GDB. Download version 7.4 (32 bits) from <http://biostatisticien.eu/springeR/32/gdb.exe> and put it in the folder `C:\Rtools\bin`. This software uses the command line and is rather austere. You may find useful to add a graphical user interface (GUI), such as the Data Display Debugger (DDD) or Emacs. Under Windows, another interesting avenue is the software Insight, included in the set of tools MinGW, available from <http://sourceforge.net/projects/mingw/files/OldFiles/insight.exe/download>. However, this software seems to be becoming obsolescent. If you try to use it, remember to change the system environment variable Path to add the path to Insight (probably `C:\insight\bin`), as explained p. 231.

Under Microsoft Windows, you will have to install the version of Emacs available at <http://vgoulet.act.ulaval.ca/en/emacs/windows>. It is a bit more complicated to use DDD under Windows. You need to launch the Cygwin



setup (available at <http://cygwin.com/install.html>), choose the installation directory C:\Rtools\bin and select the software Devel: ddd and Math: gnuplot (and accept the required dependencies). Also note that if the list of download sites is empty, you can try the URL <http://cygwin.mirrorcatalogs.com>. To use DDD, you also need an implementation of the Linux X window system for Microsoft Windows. The software Xming, available at <http://biostatisticien.eu/springeR/Xming-6-9-0-31-setup.exe>, is a good choice. You could also use MobaXterm (<http://mobaxterm.mobatek.net>), or Cygwin's Xorg server (select X11: xorg-server: X.Org servers on installation).

8.6.4 Debugging with GDB

Start an MS-DOS command window from the Windows Start menu (type cmd) in which you type

```
cd path to folder containing inv.cpp
g++ -IC:\newmat -o inv.o -c inv.cpp -g
g++ -shared -o inv.dll inv.o -IC:\newmat C:/newmat/newmat.a
```

This will create the file inv.dll with debugging information (see p. 247 for the creation of the library newmat).

Tip

In order to also debug the functions from library newmat, you need to first create this library in a way that includes debugging information:

```
cd \
cd newmat
g++ -c *.cpp -Wno-deprecated -g
ar cr newmatdebug.a *.o
ranlib newmatdebug.a
cp newmatdebug.a newmatdebug.dll
```



Then type:

```
gdb Rgui
(gdb) run
```

This should start R, where you type

```
> setwd("path to file inv.dll")
> dyn.load("inv.dll")
```

Then go to menu **Misc/Break to debugger**, which will allow you to return to GDB (black window), where you can type

```
(gdb) info share
(gdb) break inv.cpp:1
(gdb) signal 0
```

The first instruction (`info share`) shows that the library `inv.dll` has been loaded; the second instruction (`break inv.cpp:1`) allows you to add a break point on the first (executable) line of the file `inv.cpp`; the last instruction (`signal 0`) exits GDB and returns to R. In R, type:

```
> A <- matrix(rnorm(4), nrow=2)
> source("inv.R") # File created page 247.
> inv(A)
```

When the processor encounters the break point, the code execution is suspended. You can now type the following instructions in GDB. The first instruction (`list`) displays the next lines to execute, the second instruction (`next`) moves to the next line, the third instruction (`print nrow[0]`) displays the value of `nrow[0]` and the last instruction continues the code execution until the end or the next break point.

```
(gdb) list
(gdb) next
(gdb) print nrow[0]
(gdb) continue
```

You are back in R and you see the output of the call `inv(A)`. You can type the following instructions to verify that the result is the same as with function `solve()` and to exit R.

```
> solve(A)
> q()
```

Linux

Under Linux, type in a terminal window the command

`R -d gdb`

instead of `gdb Rgui`.

Alternatively, you could use the following instructions:

```
export R_HOME=/usr/lib64/R
gdb /usr/lib64/R/bin/exec/R
```

To return to R from GDB, use the key combination `CTRL+C`. Note that to go from GDB to R, after typing `signal 0` (or equivalently `c`), you need to press `RETURN`.



Tip

Note that GDB can be called with options. For example,

--directory=DIR Search for source files in DIR.
--pid=PID Attach to running process PID.

**See also**

The documentation of GDB, available at <http://sourceware.org/gdb/current/onlinedocs/gdb>, is worth reading.

**Tip**

You can install/compile a package (hereafter called PKG) with debugging information (equivalent to using the flag `-g` mentioned above). First create a file called `Makevars.win` (`Makevars` under Linux) in a subfolder called `.R/` in your `%HOME%` directory. This file should include the following lines:

```
## for C++ code
CXXFLAGS=-g
```

For this purpose, you can for example type `WINDOWS+R`, `cmd`, `ENTER`, `cd %HOME%`, `ENTER`, `mkdir .R`, `ENTER`, `cd .R`, `ENTER`, `echo CXXFLAGS=-g > Makevars.win`, `ENTER`. Next, build the package PKG and install it (from the sources using the command `R CMD INSTALL --build --debug PKG`), then use one of the debugging methods presented above. Note that the file `NAMESPACE` of your package PKG must include the line `useDynLib("PKG")` so that the DLL (or `.so`) file is automatically loaded when you execute in R the instruction `require("PKG")`. If this procedure fails, you can always use the function `dyn.load()` to load the package “by hand” from where it is installed.

**Tip**

It is also possible to display the contents of an object of type `SEXP` (call this object `s`). To do this, you can include in your C/C++ code the instruction `PrintValue(s);`. This way, when the instruction is encountered during code execution, the contents of the object `s` will be displayed in the R console. Another solution is to use the instruction `p Rf_PrintValue(s)` from the GDB console. Note that in this case, the display of object `s` in the R console may be delayed until R takes over from GDB.



8.6.4.1 Debugging with Emacs

We have seen how to debug code with GDB. We shall now show how to perform the same kind of operations with the combination of **Emacs** (and its excellent module ESS, *Emacs Speaks Statistics*) and GDB. Note that you need to have installed GDB as explained in Sect. 8.6.3. Note also that you need to create, from an MS-Dos window, the file `combn.dll` with debugging information (flag `-g`), thanks to the following instructions:

```
g++ -g -c combn.cpp -o combn.o
g++ -shared -o combn.dll combn.o
```

Note



Under **Emacs**, the notation `M-x` means you must press simultaneously the keys ALT and X, whereas `C-x` means you must press simultaneously the keys CTRL and X, and `[RET]` designates the carriage return (key RETURN).

First open **Emacs** (see p. 258 for how to install this software) then execute the following commands. For example, the first line is executed by pressing simultaneously on ALT and X, then R (which will display `M-x R` at the bottom of **Emacs**), then RETURN (which will display `ESS [S(R): R (newest)] starting data directory? ~/`), then RETURN again (which will start **R** in **Emacs**).

```
M-x R [RET] [RET]
M-x gdb [RET] gdb -i=mi --annotate=3 [RET]
```

Your **Emacs** window should then be split in two, with **R** on top and **GDB** at the bottom. If that is not the case, go to the menu **File/Split Window** or **File/New Window Below** (`C-x 2`), then to the menu **Buffer** to select `*R*`.

Warning



The system environment variable `Path` must include the entry `C:\Rtools\bin` first, so that the version of **GDB** used is 7.4.

You then need the process ID of **R**. Under Windows, use the key combination **CTRL+ALT+Del** to start the task manager. Then select the Processes tab. In the menu **View>Select Columns...**, tick the box `PID (Process Identifier)`, which will add a column `PID` to the task manager. Then find the (PID) corresponding to the name `Rterm.exe *32` (e.g., 5404). An easier option is to type `Sys.getpid()` in the upper **R** windows of **Emacs**.

Linux



Under Linux, you can get the PID of **R** directly by typing in **Emacs**:

```
M-! Shell command: pgrep R [RET]
```

Then type in **Emacs** the following instructions:

```
(gdb) attach 5404 [RET]
(gdb) signal 0 [RET]
```

Click on the panel (or *Buffer* in **Emacs**) called *R*, and execute the following instructions:

```
> setwd("path to combn.R file")
> source("combn.R")
> dyn.load(paste("combn", .Platform$dynlib.ext, sep = ""))
```

Click on the bottom sub-window (*Buffer* *gud*).

```
C-c C-c
(gdb) b combn.cpp:1 [RET]
(gdb) c [RET]
```

Click on the top sub-window (*Buffer* *R*).

```
> combnRC(5, 3)
```

```
C-g
M-x gdb-many-windows
```

Put the **Emacs** window in full screen. Your **Emacs** window should now be divided in six panels, as shown in Fig. 8.2. If needed, click on the relevant entries of the **Buffer** menu.

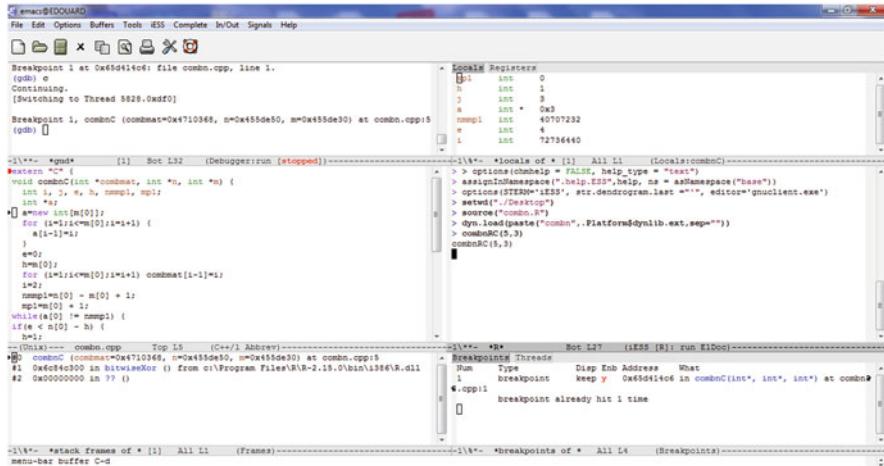


Fig. 8.2: Emacs and GDB

Click on the bottom right panel called *breakpoints of*. Select the menu **Buffers/R***.

Now click on the window **combn.cpp**. You will see new icons in the top part of **Emacs**. For example, you can click on the symbol for **Next Line** (right of G0) to execute your C/C++ line by line.

Do it yourself

- Change line 32 of file integ.cpp into `limit[0] = -1;`. Recompile this code and call it from R as seen above: `.C("testintegral",val = 0.0)$val`. Your R session should crash. Suppose you do not remember making the above change. Use the techniques you just learnt to find the error.
- Debug the file pmvtp.cpp seen in Sect. 8.5.4. Type the instruction `p Rf_PrintValue(Rpmvtp)` from the GDB console to display (in the R console) the contents of object `Rpmvtp`.

8.6.4.2 Debugging with DDD

You first need to launch Xming (or an equivalent tool); its icon should appear in the task bar. Then launch a Cygwin terminal window , and type the following instructions:

```
$ export DISPLAY=localhost:0.0
$ cd path to directory containing the source and DLL files
$ ddd Rgui
```

You may need to wait a while before DDD starts.

Linux


Under Linux, replace the last instruction with the command `R -d ddd`.

Next, type the following instructions in GDB (lower panel):

```
(gdb) dir $cwd
(gdb) run
```

The first instruction tells GDB to search for source files in the current directory (which would be given by the command `pwd`), thus avoiding issues due to path management in Windows. The second instruction starts R (you could also tick the box: `Program/Run in Execution Window`, and click on `Program/Run`, then on `Run`); type in R:

```
> dyn.load("inv.dll")
```

Note that the file `inv.dll` was created with debugging information, as mentioned page 259. Now go to menu `Misc/Break to debugger` to return to DDD. Go to menu `File/Open source...` and open file `inv.cpp`. Also tick the entry `Data Window` in menu `View` (and possibly entry `Display Local Variables` in menu `Data`, if you

are patient!). You can then put one or several breaking points in the code to debug (by double-clicking at the beginning of the line or by right-clicking), for example, at the instruction `M << values;`. This has the effect of displaying a stop symbol. Then type `continue` (or just `c`) in the lower part (gdb). This returns to R, where you type

```
> A <- matrix(rnorm(4), nrow=2)
> source("inv.R") # File created page 247.
> inv(A)
```

When the (first) breaking point is encountered by the processor, code execution is suspended. You can now use the graphical tool DDD to debug your code.

Note that it is possible to display several values of an array. For example, you can type in the lower window (gdb) the following instruction (Fig. 8.3):

```
graph display values[0] @ 4
```

to display the (first) four values of array `values`.

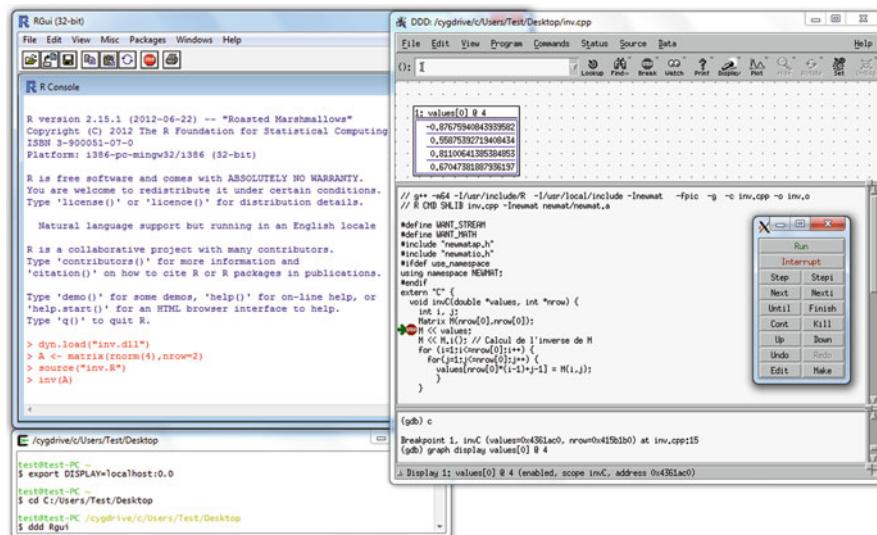


Fig. 8.3: DDD and GDB

8.6.4.3 Debugging with Insight

Insight seems to have difficulties working on some Windows versions. Nonetheless, we present this software for those who have a compatible version of Windows, or in case a new version of **Insight** is shipped after the publication of this book.

Recompile your file using flag `-g` (and possibly `-fPIC`) which tells the C++ compiler to add information on the source code directly in the compiled file.

```
g++ -c combn.cpp -o combn.o -g
g++ -shared -o combn.dll combn.o
```

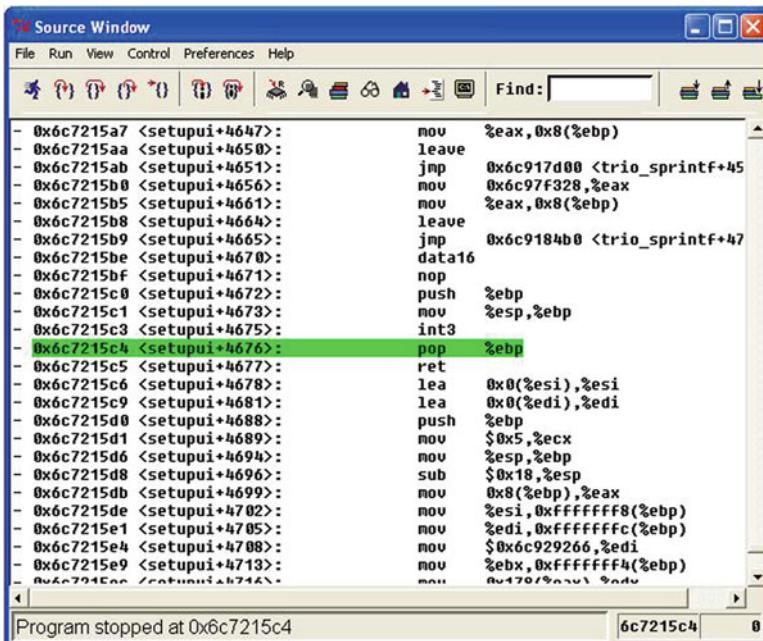
Then, from the MS-DOS window, execute `insight Rgui.exe`, then click on Run



Next type the following commands in the R console which opens:

```
> source("combn.R")
> dyn.load(paste("combn", .Platform$dynlib.ext, sep = ""))
```

Go to the R menu called **Misc**, then **Break to debugger**. You are now in the Insight window.



In Insight, select menu **View - Console** [CTRL+N]. This opens the command window of debugger GDB. We can now add a breaking point to function `combnC` by typing

```
break combnC
```

Then type:

```
continue
```

which returns to R. As soon as the function `combnC` is called, we will return to the debugger.

```
?> Console Window
(gdb) break moncombn
Breakpoint 1 at 0x20c11d6: file moncombn.cpp, line 7.

(gdb) continue
Continuing.
gdb: child_resume.SetThreadContext: thread 3408.0xd98
ContinueDebugEvent (cpid=3408, ctid=3480, DBG_CONTINUE);
```

Now type in R:

```
> debug(combnRC)
> combnRC(5, 3)
```

Use instruction n (for *next*) to skip to the next instruction of our R code, until reaching the call to the function written in C++.

```
?> RGui
Fichier Edition Voir Misc Packages Fenêtres Aide
[Icons]
?> R Console
> setwd("C:\\\\Documents and Settings\\\\iafaye\\\\Bureau")
> source("moncombn.R")
> dyn.load(paste("moncombn", .Platform$dynlib.ext, sep = ""))
> debug(moncombn)
> moncombn(5, 3)
debugging in: moncombn(5, 3)
debug: (
  combmat <- matrix(0, nrow = m, ncol = choose(n, m))
  dyn.load(paste("moncombn", .Platform$dynlib.ext, sep = ""))
  out <- .C("moncombn", res = as.integer(combmat), as.integer(n),
    as.integer(m))
  combmat <- matrix(out$res, nrow = m, byrow = F)
  dyn.unload(paste("moncombn", .Platform$dynlib.ext, sep = ""))
  return(combmat)
)
attr(,"srcfile")
moncombn.R
Browse[1]> n
debug: combmat <- matrix(0, nrow = m, ncol = choose(n, m))
Browse[1]> n
debug: dyn.load(paste("moncombn", .Platform$dynlib.ext, sep = ""))
Browse[1]> n
debug: out <- .C("moncombn", res = as.integer(combmat), as.integer(n),
  as.integer(m))
Browse[1]> |
```

The breaking point we added is detected and we are back in Insight.

```

1 // Fonction moncombn
2 extern "C" {
3 void moncombn(int *compmat, int *n, int *m)
4 {
5   int i, j, e, h, nmmp1, mp1;
6   int *a;
7   a=new int[*(*n+0)];
8   for (i=1;i<=*(*n+0);i=i+1) *(a+i-1)=i;
9   e=0;
10  h=*(m+0);
11  for (i=1;i<=*(*m+0);i=i+1) *(compmat+i-1)=i;
12  i=2;
13  nmmp1=*(n+0) - *(m+0) + 1;
14  mp1=*(m+0) + 1;
15  while(*a+0 != nmmp1) {
16    if(e < *(n+0) - h) {
17      h=1;
18      e=*(a+*(m+0)-1);
19      *(a+*(m+0) - h)=e + 1;
20      for (j=1;j<=*(*n+0);j=j+1) *(compmat+(i-1)**(n+0)+j-1)=*(a+j-1);
21      i=i+1;
22    }
23  else {
24    h=h + 1;
25    e=*(a+mp1 - h-1);
26    for (j=1;j<=h;j=j+1) *(a+*(m+0) - h + j-1)=e + j;
27    for (i=1;i<=*(*m+0);i=i+1) *(compmat+(i-1)**(m+0)+i-1)=*(a+i-1);
}
}
}

```

Program is running. 20c11d6 7

Next click on icon to execute line by line the C++ code and check the value of the various variables.

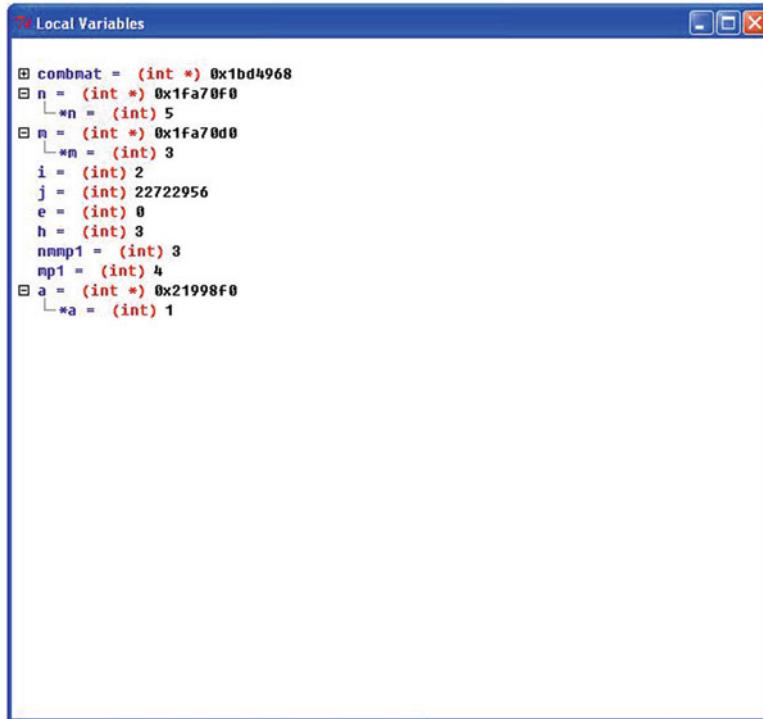
```

1 // Fonction moncombn
2 extern "C" {
3 void moncombn(int *compmat, int *n, int *m)
4 {
5   int i, j, e, h, nmmp1, mp1;
6   int *a;
7   a=new int[*(*n+0)];
8   for (i=1;i<=*(*n+0);i=i+1) *(a+i-1)=i;
9   e=0;
10  h=*(m+0);
11  for (i=1;i<=*(*m+0);i=i+1) *(compmat+i-1)=i;
12  i=2;
13  nmmp1=*(n+0) - *(m+0) + 1;
14  mp1=*(m+0) + 1;
15  while(*mp1!=0) { // Line 15 highlighted in green
16    if(e < *(n+0) - h) {
17      h=1;
18      e=*(a+*(m+0)-1);
19      *(a+*(m+0) - h)=e + 1;
20      for (j=1;j<=*(*n+0);j=j+1) *(compmat+(i-1)**(n+0)+j-1)=*(a+j-1);
21      i=i+1;
22    }
23  else {
24    h=h + 1;
25    e=*(a+mp1 - h-1);
26    for (j=1;j<=h;j=j+1) *(a+*(m+0) - h + j-1)=e + j;
27    for (i=1;i<=*(*m+0);i=i+1) *(compmat+(i-1)**(m+0)+i-1)=*(a+i-1);
}
}
}

```

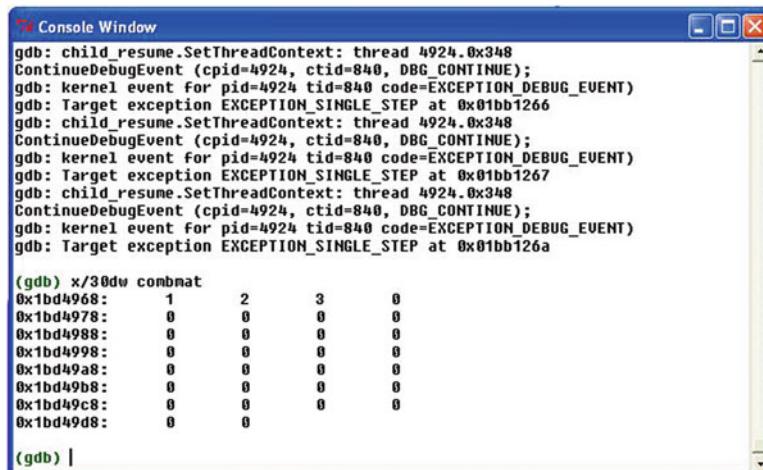
Program stopped at line 15. 20c126a 15

The window **Local Variables** (shown by menu **View -> Local Variable** [CTRL+L]) displays all local variables and their contents during code execution.



Note that to see the contents of an R matrix or vector, you simply need to go to the GDB console and type for example:

```
x/30dw combmat
```



You can also display graphically this table of values and select it by clicking on plot.

You can now type the following instructions in the GDB console to add a breaking point at line 32 of your C++ code, then reexecute the code. When the breaking point is encountered, the code stops again and we can ask to display again the contents of array x:

```
break 32
continue
x/30dw combmat
```

```
? Console Window
(gdb) break 32
Breakpoint 2 at 0x1bb13be: file moncombn.cpp, line 32.

(gdb) continue
Continuing.
gdb: child_resume.SetThreadContext: thread 4924.0x348
ContinueDebugEvent (cpid=4924, ctid=840, DBG_CONTINUE);
gdb: kernel event for pid=4924 tid=840 code=EXCEPTION_DEBUG_EVENT
gdb: Target exception EXCEPTION_BREAKPOINT at 0x01bb13be

Breakpoint 2, moncombn (compmat=0x1bd4968, n=0x1fa70f0, m=0x1fa70d0) at moncom

(gdb) x/30dw combmat
0x1bd4968:    1      2      3      1
0x1bd4978:    2      4      1      2
0x1bd4988:    5      1      3      4
0x1bd4998:    1      3      5      1
0x1bd49a8:    4      5      2      3
0x1bd49b8:    4      2      3      5
0x1bd49c8:    2      4      5      3
0x1bd49d8:    4      5

(gdb) |
```

8.6.4.4 Detecting Memory Leaks

The messages Segmentation fault (or segfault), invalid next size, std::bad_alloc (which you will certainly encounter under Linux!), incoherent results or, more radically, a complete crash of R are often indications that there is a memory issue (access to a non-reserved or non-initialized address, using freed memory, etc.) These memory leaks often occur when you have forgotten to use the instruction `delete[] ptr;` to delete from memory a pointer `ptr` introduced in a C/C++ function. This problem can sometimes be noticed in the task manager when you run a large simulation in R and realize that the R process is using more and more memory even though it should not.

Linux



Under Linux, the display of memory usage by different processes is given by the command (entered in a terminal window) `watch -d free` for global usage or by `top -p PID` for a specific process (use `ps au` to find the PID of the desired process). You can also use the graphical tool `ksysguard`.

Another common mistake is to try to manipulate the n th entry in an array of size less than n (accessing undefined memory). It can then be difficult to detect the origin of the problem. The software Dr Memory (<http://code.google.com/p/drmemory>) and possibly the software electric-fence-win32 (<http://code.google.com/p/electric-fence-win32>) and duma (<http://duma.sourceforge.net>) can be precious tools in such situations.

Linux

Under Linux, you can use the software Valgrind or Electric Fence.



We now show on an example how to use Dr. Memory which you should install in the directory C:\drmemory (choose the entry Add Dr. Memory to the system PATH for all users upon installation).

The following piece of code includes several errors, which can be hard to find for a beginner. You can download it from <http://biostatisticien.eu/springer/memory.cpp>.

```
1 extern "C" {
2     void testmemory( int *M, double * a ) {
3         double * ptr1 , * ptr2 ;
4         int i ;
5         ptr1 = new double[10000];
6         ptr2 = new double[M[0]];
7         ptr1 [0] = 1.0;
8         for ( i=1;i<10000;i++ ) {
9             ptr1 [i] = ( double )i ;
10            ptr2 [i] = ptr1 [i-1] * ( double )i ;
11        }
12        delete [] ptr2 ;
13        for ( i=0;i<10;i++ ) a[ i ] = ptr2 [ i ];
14        return ;
15    }
16 }
```

First create the associated DLL file, using the following instructions in an Ms-Dos window:

```
cd directory containing file memory.cpp
g++ -o memory.o -c memory.cpp -g
g++ -shared -o memory.dll memory.o
```

Linux

Under Linux, use the instructions:



```
g++ -o memory.o -c memory.cpp -g -fPIC
g++ -shared -o memory.so memory.o
```

Next, type `drmemory.exe -- Rgui` in your command window (be patient), then type the following instructions in the R console:

```
> dyn.load("memory.dll")
> .C("testmemory",10000L,3.0)
> q()
```

Now look for the instances of `testmemory` in the file which opened up. This will indicate the lines which may contain errors. For example, this shows that there is an error at line 13. In fact, we realize that the array `a` is of length 1 (and initially contains only the value 3.0), whereas we are trying to write values in entries 0–9. Furthermore, the pointer `ptr2` was deleted on the preceding line.

You can also try the following R instruction, and note in the task manager that the amount of RAM used by R increases greatly. This is because we forgot the instruction `delete[] ptr1;` in the C/C++ code above:

```
> for (i in 1:10000) .C("testmemory",10000L,as.double(1:10))
```

Linux

The equivalent of Dr Memory under Linux is called Valgrind. To detect where the leak comes from, you can use the instruction:

```
R -d 'valgrind --leak-check=full'
```

```
> dyn.load("memory.so")
> .C("testmemory",10000L,3.0)
> q()
```



In the output of valgrind, you then need to look for the errors and for the corresponding line numbers in the source code of `memory.cpp`. The following instructions give other error types displayed by R and detected by Valgrind:

```
> # Works only once!
> # Afterwards, R crashes with: "caught segfault":
> .C("testmemory",10000L,c(3.0,5.0))
> # R closes: "invalid next size":
> .C("testmemory",10000,c(3.0,5.0))
> # R closes: "std::bad_alloc":
> .C("testmemory",10^12,c(3.0,5.0))
> # Works when ptr2 is no longer defined:
> .C("testmemory",10000L,as.double(1:10))
```

SECTION 8.7

Parallel Computing and Computation on Graphical Cards

8.7.1 Parallel Computing

You can speed up your calculations by having them run on several processors at the same time; these processors can even be on different computers. There are several specialized packages for this; they are listed in the CRAN Task View: High-Performance and Parallel Computing with R, available at <http://cran.r-project.org/web/views/HighPerformanceComputing.html>.

The easiest to use is package `parallel` with communication protocol PSOCK, which we briefly describe below through an example.

Tip

The MPI protocol (*Message Passing Interface*), used by package `Rmpi`, is more flexible than the PSOCK protocol, but it requires the installation of other software (such as OpenMPI or `mpich2`).



See also

We refer the interested reader to the websites <http://www.divms.uiowa.edu/~luke/R/cluster/cluster.html>, <http://www.sfu.ca/~sblay/R/snow.html> and <http://cran.r-project.org/web/packages/snowfall/vignettes/snowfall.pdf>.



The following R code performs numerical evaluation (by Monte Carlo simulation) of the empirical level of the Shapiro-Wilks normality test for a nominal level of 5 %:

```
> myfunc <- function(M=1000) {
+   decision <- 0
+   for (i in 1:M) {
+     x <- rnorm(100)
+     if (shapiro.test(x)$p < 0.05) decision <- decision + 1
+   }
+   return(decision)
+ }
```

Here is the computation time needed for this code with $M = 60,000$ Monte Carlo iterations:

```
> system.time({
+   M <- 60000
```

```
+ decision <- myfunc(M)
+ print(decision/M)
+ })
[1] 0.04893333
  user  system elapsed
18.124   0.331 18.457
```

We now show how this code can be parallelized using the package `parallel` and the corresponding gain in computation time. We used six processors.

Tip

To know the number of processors on your computer, type the instruction `devmgmt.msc` in the menu Start/Run. Then count the number of lines in the Processors entry. Under Linux, type `top` in a terminal window, then 1. This shows the number of processors. Another option is to use the function `detectCores()` of package `parallel`.

```
> require("parallel")
> system.time({
+ nbclus <- 6
+ M <- 60000
+ cl <- makeCluster(nbclus, type = "PSOCK")
+ out <- clusterCall(cl, myfunc, round(M/nbclus))
+ stopCluster(cl)
+ decision <- 0
+ for (clus in 1:nbclus) {
+   decision <- decision + out[[clus]]
+ }
+ print(decision/(round(M/nbclus)*nbclus))
+ })
[1] 0.0501
  user  system elapsed
 0.019   0.033   5.522
```

8.7.2 Computation on Graphical Cards

The processor, or CPU (*central processing unit*), is the computer component which handles execution of software. However, it is now also possible to perform computations on a GPU (*graphical processing unit*), or graphical card. Graphical cards allowing such operations are marketed by Nvidia, and they can include hundreds of processors working in parallel. The speed-up in computation time can be substantial. To use this technology, however, you must know the programming language CUDA, developed by Nvidia. A few R developers have delved into this language and have grouped a few functions in the package `gputools`, which is only available on Linux for now.

Here is a short example of use of this package. We used an NVIDIA GeForce GTX 480 graphical card.

```
> require("gputools")
> A <- matrix(runif(40000), nrow=200, ncol=200)
> B <- matrix(runif(40000), nrow=200, ncol=200)
> system.time(cor(A, B, method="kendall")) # Computation CPU.
  user  system elapsed
29.804    0.002 29.810
> system.time(gpuCor(A, B, method="kendall")) # Computation on
                                                # GPU.
  user  system elapsed
0.836    0.052   0.891
```

See also

To find out more on this topic, go to <http://cran.r-project.org/web/packages/gputools/gputools.pdf> and http://developer.nvidia.com/object/cuda_training.html.



Memorandum

```

function(<par1>, <par2>, ..., <parN>) <body> : declare a function object
"{"(): define a block of instructions and return the last evaluated instruction
class(), "class<-"(): extract, affect the class of an object
missing(): test whether an effective argument is defined
attributes(), "attributes<-"(): extract, affect all attributes as a list
attr(), "attr<-"(): extract, affect a single attribute
expression(): create an expression object
parse(): convert text to an expression
eval(): evaluate an expression
"~"(): create a formula object
new.env(): create an environment
local(): execute code locally in an environment

```



Exercises

8.1- For each of the following command lines, indicate the class of the returned R object. What is displayed upon execution of each of these command lines?

- `function(name) {name}`
- `(function(name) {name})(“Ben”)`
- `(function(name) {cat(name, “\n”)})(“Ben”)`
- `(function(name) {invisible(name)})(“Ben”)`

8.2- Is there a difference between

- `name <- function(name) name` and `name <- function(name) {name}`
- `name <- function(name) {name}` and `name <- function(name) {return(name)}`
- `name <- function(name) {name}` and `(function(name) {name}) -> name`

8.3- Upon execution, is there a difference between `name()` and `name(“Peter”)` when

- `name <- function(name=“Peter”) name`
- `name <- function(name=“Peter”) name2 <- name`

For these two declarations of the function `name()`, is there a difference in the type of the R object `res` given by `res <- name(“Ben”)`?

8.4- What R object is returned upon execution of `name()` when

```

name <- function(name=“Peter”) {
  name
  # The last instruction is a comment!
}

```

- 8.5-** When `name <- function(firstname="Peter", name="L") {
 paste(firstname, name)}`, what R object is returned by

- `name(firstname="Ben")`
- `name(fir="Ben")`
- `name(n="D", f="R")`

- 8.6-** Rewrite the following function declaration in one line, without using the command separator “;”:

```
name <- function(name) { if(missing("name"))
  name <- "Peter"; cat(name, "\n") }
```

- 8.7-** What is the output of the execution of `nameS("peteR", "Ben", "R")` when

- `nameS <- function(...) c(...)`
- `nameS <- function(...) list(...)`
- `nameS <- function(...) for(name in c(...)) print(name)`
- `nameS <- function(...) for(name in list(...))
 print(name)`

Same question upon execution of

```
nameS(c("peteR", "L"), c("Ben", "L"), c("R", "D"))
```

- 8.8-** When `nameS <- function(names=c("Ben", "R"), ...) c(names, ...)`, which R objects are returned by `nameS("PeteR")`, `nameS(name="PeteR")` and `nameS(names="PeteR")`? Same question when
`nameS <- function(..., names=c("Ben", "R")) c(names, ...)`.

- 8.9-** Create a constructor function `Male()` generating an object of class "Male" with fields `firstname` and `name` (in an object of type `list`). Create the method `hello.Male()` which displays "Hello Mister FIRSTNAME NAME!" (do not forget the "\n" at the end of the display!) for an object with values "FIRSTNAME" and "NAME", respectively, for the fields `firstname` and `name`. When `man <- Male("Ben", "L")`, what is produced upon execution of the following commands: `hello.Male(man)` and `hello(man)`? What code should you execute in addition for the two results to be identical?

- 8.10-** Create the analogous functions for the class "Female" (hint: do not forget to update the gender in `hello.Female()`). When
`woman <- Female("Elsa", "R")`, what is produced upon execution of the following commands: `hello.Male(woman)`, `hello.Female(woman)` and `hello(woman)`?

- 8.11-** When `welcome <- function(...) for(person in list(...)){
 hello(person)}`, what is returned by `welcome(man, woman)`?

And when `welcome <- function(...) for(person in c(...)){
 hello(person)}`?

Same question when `hello.default <- function(obj){
 cat("hello", obj, "!\\n")}`.



Worksheet

Programming Functions and Object-Oriented Programming in R

Before reading the practicals of this chapter, we strongly advise you to revise those of the previous chapters (especially the one on “advanced plots”) and to reorganize their solutions in as many functions as necessary.

A- Managing a Bank Account

The aim of this practical is to create three minimalist functions to manage bank accounts. The accounts will be stored in data.frame objects all called `accounts` and stored in different .RData files. All these files will be located in the same folder. The path to this folder should be saved in the R variable `.folder.accounts` and be accessible in all the functions you develop.

- 8.1-** The instruction `file.path(.folder.accounts, paste(name, ".RData", sep=""))` gives the path of the file associated with the account `Name`. Create the functions `path.account()`, which takes one formal argument `name` (representing the name of the account) and returns the complete path to the file (which contains the object `account` of class `data.frame`) with extension `.RData`.
- 8.2-** Given that `factor(levels=c("Debit", "Credit"))`, `numeric(0)` and `character(0)`, respectively, give empty vectors of explicit types, which expression would generate an empty data matrix with the predefined fields `amount`, `mode`, `date` and `remark`? Create the function `account()` (not to be confused with the variable `account` called in its body) which takes one argument `name` and creates a new account.
- 8.3-** Create the functions `debit()` and `credit()` to, respectively, debit and credit an amount `amount` (second argument) from the account `name` (first argument). The third argument is any comment to put as `remark`. A fourth argument can represent the date; the default value is
`format(Sys.time(), "%d/%m/%Y")` (*i.e.* the date of input). Remember to use the functions `load()` and `save()` to load and save the variable `account` in the body of each function.
- 8.4-** If `account` is the data matrix containing information on the account, what is returned by `sum(account[account$mode=="Credit", "amount"])`? Modify the function `account()` so that it returns the current state of the account only when the file returned by `path.account(name)` exists (use the function `file.exists()` to test whether a file exists).
- 8.5-** Complete account management by creating any additional functions you wish.
- 8.6- Optional question:** Since most use of R is done with objects, adapt the previous functions in a way that respects the R object-oriented philosophy. You can use the next practicals for inspiration.

B- Organizing Graphical Objects

When you think about it, plots in R do not really respect the object-oriented spirit: unlike most other entities, an R plot is not considered as an object which can be saved (and possibly modified) and on which certain methods can be applied. We shall attempt to propose a very basic prototype to draw a plot with circles and rectangles (and hence squares). You can enrich this library with graphical objects as you wish. Our aim is to maintain a list of graphical objects, with the possibility of changing any of its elements at any time.

- 8.1-** R functions `plot.new()` and `plot.window()` are used to initialize a plot.

The argument `asp` set to 1 creates plots with correct units for the `x` and `y` axes. Propose an object `Window` which gives the user the option of saving the dimensions of the graphics display window. The user can then call the constructor function (or method) `Window()` (which could have the same name as the class), which takes as arguments `x` and `y` (the coordinates of the centre), `width`, `height` (dimensions along the `x` and `y` axes, respectively) and optionally `log` (logarithmic transformation). All these quantities should be stored in an object `list`, returned by the constructor function `Window()`, after affecting its class to "`Window`".

- 8.2-** Similarly, propose constructor functions for objects of classes `Circle` and `Rectangle`. The fields `x` and `y` represent the coordinates of the centre of the object, `radius` is the radius of a circle and `width` and `height` are the dimensions of a rectangle.
- 8.3-** Propose plotting methods `plot.Window()`, `plot.Rectangle()` and `plot.Circle()`. You can find inspiration in the following R treatments used to display a new plot with a circle and a square centred at the origin and of diameter and side length set to 1:

```
plot.new()
plot.window(xlim=c(-1,1),ylim=c(-1,1),asp=1)
rect(-.5,-.5,.5,.5)
symbols(0,0,circle=.5,inches=FALSE,add=TRUE)
```

- 8.4-** Test the code you have developed by executing the code:

```
mywindow <- Window(0,0,2,2)
mycircle <- Circle(0,0,.5)
myrectangle <- Rectangle(0,0,1,1)
plot(mywindow);plot(mycircle);plot(myrectangle)
```

If all goes well, you should see a graphics window with a circle inside a square.

- 8.5-** We now need to develop the methods associated with the class `MyPlot` which will contain the list of all graphical objects. First, propose a constructor function `MyPlot()` which initializes an object as `list(objects=list())` (where `objects` is the field containing the list of graphical objects), affects the class "`MyPlot`" and returns the object.

- 8.6-** Propose a method `add.MyPlot()` which adds graphical objects. Remember to give a generic function `add()` to launch all associated methods. Use the functionalities of the list of supplementary arguments ... and the function `c()` so that the method `add.MyPlot()` can initialize as many graphical objects as the user wishes. Propose a method `plot.MyPlot()` which executes the methods `plot()` for all graphical objects. The user can then enter the following lines to get the same result as earlier:

```
myplot <- MyPlot()
myplot <- add(myplot, Window(0,0,2,2), Circle(0,0,.5),
              Rectangle(0,0,1,1))
plot(myplot)
```

- 8.7-** To display a plot, you need to initialize an object of type `Window` and put it in first position of the list of graphical objects of the class `MyPlot`. It might be useful to initialize it directly inside the constructor function `MyPlot()`. The arguments of the function `Window()` can be proposed directly for the function `MyPlot()`. Another idea is to propose a list of graphical objects to the user upon creation of an object of class `MyPlot`. As we have done for the method `add.MyPlot()`, we could use the list of supplementary arguments ..., which must be placed as first argument of the function `MyPlot()` so as to get the previous result with only two lines:

```
myplot <- MyPlot(Circle(), Rectangle())
plot(myplot)
```

However, note that in the first line, it is assumed that the default values of the arguments of the function `Window()`, `Circle()` and `Rectangle()` are appropriate.

- 8.8-** The project is launched with this first prototype. You can complete it as you wish. If you need inspiration, you could try managing the list of graphical objects (e.g., deleting or modifying an object), display styles, axes, etc.

C- Creating a Class `lm2` for Linear Regression with Two Regressors

The aim of this practical is to reproduce the procedure used by our two friends for simple regression. Graphical display will be made possible by the excellent package `rgl`, which is an OpenGL interface for `R`. Given the technical difficulty of this chapter, we propose here to develop functions (actually methods). Given that some aspects are very technical, the aim is only to get the reader to understand all the development steps of the following functions. This practical is aimed at more advanced users.

The following function returns an object of class `lm2` which inherits from the standard class `lm`.

```
1 lm2 <- function (...) {
2   obj <- lm (...)
```

```

3 | if(ncol(model.frame(obj))!=3)
4 |   stop("two independent variables are required!")
5 | class(obj) <- c("lm2",class(obj)) # or c("lm2","lm")
6 | obj
7 |

```

For example, execute the following commands:

```

> n <- 20
> x1 <- runif(n,-5,5)
> x2 <- runif(n,-50,50)
> y <- 0.3+2*x1+2*x2+rnorm(n,0,20)
> reg2 <- lm2(y~x1+x2)
> summary(reg2)
Call:
lm(formula = . . .)
Residuals:
    Min      1Q  Median      3Q     Max
-32.0767 -17.1529  0.9872  12.3298  35.5909
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.8708     5.0769  -0.368   0.717
x1          2.8400     1.9594   1.449   0.165
x2          1.8084     0.1952   9.263  4.7e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 21.14 on 17 degrees of freedom
Multiple R-squared:  0.848,    Adjusted R-squared:  0.8301
F-statistic: 47.42 on 2 and 17 DF,  p-value: 1.112e-07

```

No surprises here: the R output of the summary is given by the method `summary.lm()`.

The user now wishes a 3D scatter plot with the regression plane given by the standard method of least squares.

```

1 plot3d.lm2 <- function(obj, radius=1, lines=TRUE,
2                           windowRect, . . .) {
3   matreg <- model.frame(obj)
4   colnames(matreg) <- c("y", "x1", "x2")
5   predlim <- cbind(c(range(matreg[,2]),
6                     rev(range(matreg[,2]))),
7   rep(range(matreg[,3]), c(2,2)))
8   predlim <- cbind(predlim, apply(predlim, 1,
9     function(1) sum(c(1,1)*coef(obj)))
10  ))
11  if(missing(windowRect)) windowRect=c(2,2,500,500)
12  open3d(windowRect=windowRect, . . .)
13  bg3d(color = "white")
14  plot3d(formula(obj), type="n")
15  spheres3d(formula(obj), radius=radius, specular="green")
16  quads3d(predlim, color="blue", alpha=0.7, shininess=128)
17  quads3d(predlim, color="cyan", size=5, front="lines",
18    back="lines", lit=F)

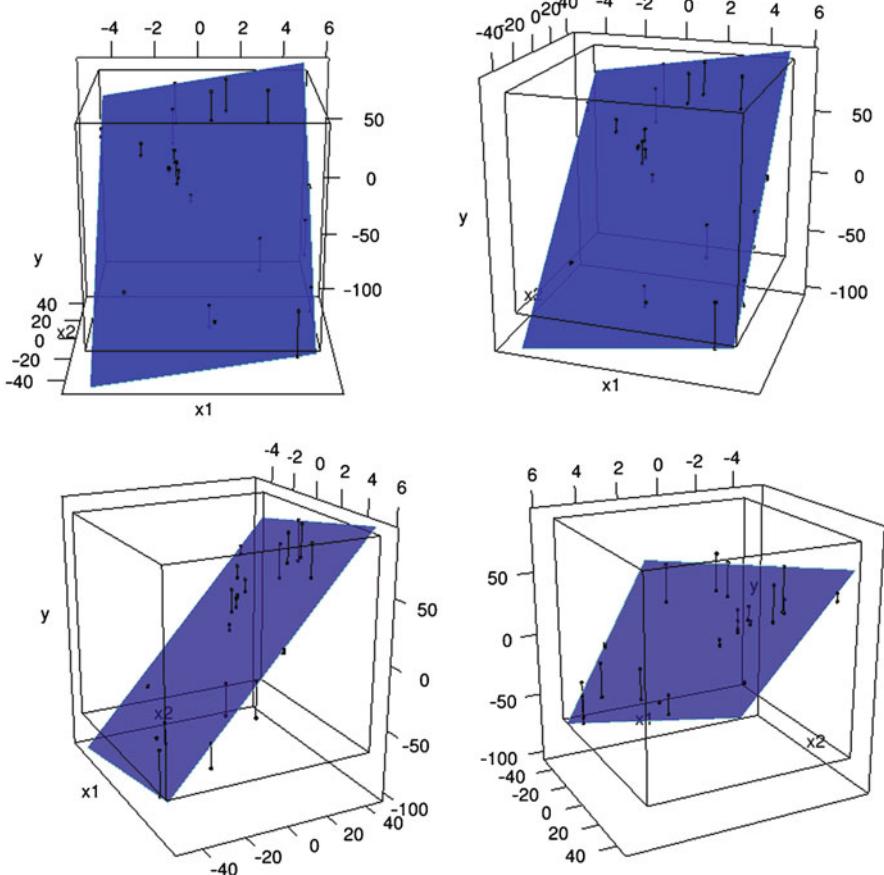
```

```

19 if(lines) {
20   matpred <- cbind(matreg[2:3],
21     model.matrix(obj) %*% coef(obj))
22   points3d(matpred)
23   colnames(matpred) <- c("x1","x2","y")
24   matlines <- rbind(matreg[,c(2:3,1)],matpred)
25   nr <- nrow(matreg)
26   matlines <- matlines[rep(1:nr,rep(2,nr))+c(0,nr),]
27   segments3d(matlines)
28 }
29 }
```

Here is a direct application of this method with four graphical illustrations for four different viewing angles.

```
> require("rgl")
> plot3d(reg2)
```



Chapter 9

Managing Sessions

Prerequisites and goals of this chapter

- Reading previous chapters.
- This chapter describes various procedures to manage R sessions. You have to follow a rather rigorous discipline and a methodology specific to R to **make sure you save your work efficiently**. We present the commands to save your work: **objects** you have created, **instructions** you have typed, **plots** you have drawn. We also present a few other useful commands and offer a short introduction to package creation.

SECTION 9.1

R Commands, Objects and Storage

• Storing objects

The basic commands are either expressions or affectations using the arrow <- or ->. If an expression is typed in, it is evaluated, and the result is displayed and then lost. An affectation also evaluates an expression, but does not necessarily display the result. The result is then stored in an object.

```
> 2*9 # The result is displayed then lost.  
[1] 18  
> My.Weight <- 75 ; My.Height <- 1.90 # These two results are  
# stored. They can be  
# re-used.  
> My.BMI <- My.Weight/My.Height^2
```

```
> My.BMI
[1] 20.77562
```

• Listing objects

After you have created R objects, you can get the list of all objects with the function `ls()` or the synonymous function `objects()`.

```
> ls()
[1] "A"          "B"          "cl"         "clus"
[5] "combnRC"    "combnRCF"   "corr"       "decision"
[9] "e"          "inv"        "lm2"        "lsq"
[13] "M"          "My.BMI"     "My.Height"  "My.Weight"
[17] "myfunc"    "n"          "nbclus"    "out"
[21] "plot3d.lm2" "pmvtrCR"   "reg2"      "space"
[25] "space2"    "space3"    "x1"        "x2"
> objects()
[1] "A"          "B"          "cl"         "clus"
[5] "combnRC"    "combnRCF"   "corr"       "decision"
[9] "e"          "inv"        "lm2"        "lsq"
[13] "M"          "My.BMI"     "My.Height"  "My.Weight"
[17] "myfunc"    "n"          "nbclus"    "out"
[21] "plot3d.lm2" "pmvtrCR"   "reg2"      "space"
[25] "space2"    "space3"    "x1"        "x2"
```

• Deleting objects

To delete objects, use the function `rm()`.

```
> rm(My.Height) # Delete the object My.Height.
> ls()
[1] "A"          "B"          "cl"         "clus"
[5] "combnRC"    "combnRCF"   "corr"       "decision"
[9] "e"          "inv"        "lm2"        "lsq"
[13] "M"          "My.BMI"     "My.Weight" "myfunc"
[17] "n"          "nbclus"    "out"        "plot3d.lm2"
[21] "pmvtrCR"   "reg2"      "space"     "space2"
[25] "space3"    "x1"        "x2"
> rm(list=ls()) # Delete all objects in the current work
# environment.
> ls()
character(0)
```

Advanced users

You can use regular expressions to delete objects following a certain name pattern. For example, the following instruction only deletes objects with a name of the form `a?b`, where `?` represents a single character:

```
rm(list=ls(pattern=glob2rx("a?b")))
```

We will not give further details and refer the interested reader to the online help for the function `glob2rx()`.

SECTION 9.2

Workspace: .RData Files

When working with R, objects are created: vectors, matrices, functions, etc. These objects are physically saved in a file on the hard disk called **workspace**. The file name extension must be **.RData** (or **.rda** in older versions of R).

It is possible (and highly advisable) to create several **.RData** files: one for each project you are working on. You should create these **.RData** files in different appropriate folders. For example, suppose you are working on two different statistical projects: one related to cars and one related to climate events. You could then create a folder called **Cars** containing a file **cars.RData** and another folder called **Climate** containing a file called **climevt.RData**; these files will contain the R objects corresponding to the two studies.

The function **save.image()** is used to save a workspace; you can use the function **load()** to load an existing workspace. Under Microsoft Windows, you can access **.Rdata** files from the menus **File/Save workspace...** and **File/Load workspace**.

Mac

Under MacOS, you can access **.Rdata** files from the menus **Workspace/Save workspace file** and **Workspace/Load workspace file**. The menu **Workspace** is also useful to explore the contents of the workspace (you can open a window with a list of all objects including their type and dimension) and to edit objects.



Note that the function **save()** will save only the objects you choose in the workspace.

Note

There is a default workspace in R. To find the path to the folder where it is stored, type the instruction **getwd()** just after launching R.



It is worth noting that the command **getwd()** returns the current working directory. The command **setwd()** is used to change working directory.

Do it yourself



Start R, then type

```
x <- 3 # Assigning a value to x; check the contents of x.
x <- 4 # Assigning a new value to x, overwriting the old value.
```

Now, outside of R, create two folders in the same directory: one called Cars, the other Climate. Then type into R the following instructions:

```
rm(list=ls()) # Start by deleting all objects
               # in the current workspace.
ls()          # Returns character(0), indicating that no
               # objects are left.
x<-c("FIAT","VOLVO","RENAULT","PEUGEOT") # Assign a value to x.
ls()          # Return x.
setwd("/path/to/Cars") # Move to folder Cars.
save.image("cars.RData") # Create the file cars.RData
                       # in the folder Cars.
```

You have created an object called x, containing a list of car manufacturers. This object is saved (in binary form) in the workspace cars.RData in the folder Cars.

Now, type the following instructions:

```
# Affect a new value to x:
x <- c("storm","hurricane","tornado","typhoon")
setwd("/path/to/Climate") # Move to folder Climate.
save.image("climevt.RData") # Create the file climevt.RData
                           # in the folder Climate.
```

You have created an object called x containing names of climate events. Note that the old value of x has been overwritten in the current workspace. This new object x is saved in the workspace climevt.RData in the folder Climate. Exit R (you can use the function q("yes")). Start R again and type in the following instructions:

```
ls()          # Returns x; check the value of x
load(file.choose()) # Open the file cars.RData
                   # in the folder Cars.
ls()          # Returns x; check the value of x.
load(file.choose()) # Open the file climevt.RData
                   # in the folder Climate.
ls()          # Returns x; check the value of x.
q("no")       # Exit R.
```

This shows the point of having several workspaces: you can keep several objects which have the same name but contain different information. Otherwise, the second affectation of a value to x would overwrite the first!

Tip

When exiting R with the command `q()` (or by clicking the cross at the top right corner of the R window under Windows, or on the red button at the top left corner for Macintosh users), the following question is asked:

Save an image of the session?

If you answer Yes (or `y` under Linux), a workspace file called `.RData` (containing all objects created during the session) and a command history file called `.Rhistory` (which we explain in the next section) are saved in the current directory.

**Note**

The function `attach()` plays a similar role to the function `load()`. The difference between these two functions is explained later on.

**SECTION 9.3**

Command History: .Rhistory Files

R includes a mechanism to recall and reexecute old commands. The up and down arrows on the keyboard can be used to go back and forward in the command history. Once you have located a command using this method, you can move the cursor using the right and left arrows, delete characters with the DEL key and add or modify characters with the keyboard.

Just like all objects can be saved in a dedicated file with the command `save.image()`, you can also save all commands you typed. The commands are saved in a file which must have the extension `.Rhistory` (or `.rhi` in old versions of R).

Here again, it is a good idea to save a `.Rhistory` file for each project you are working on. You can then access these commands in an interactive fashion from the R command line, using the keyboard arrows.

To save the command history of the current session, use the command `savehistory()`. To load the command history from a previous session, use the command `loadhistory()`. Under Microsoft Windows, the same operations can be done from the menus `File/Save History...` and `File/Load History...`

Mac

Mac users can rely upon R.app which offers a lateral bar to view, navigate and manipulate the history. It is activated by clicking the icon Show/Hide history in the R console.

Tip

The command `history()` opens a new window with a list of all commands from the current session.

Do it yourself

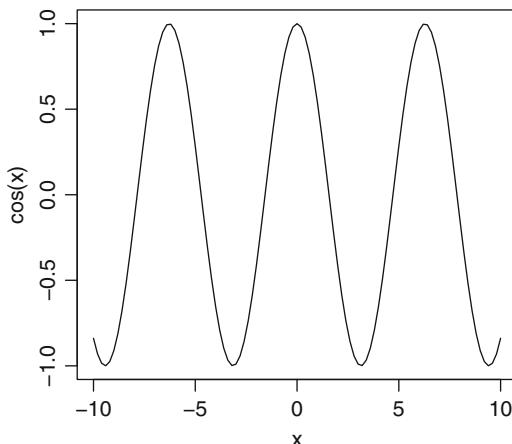
Start R and type in the following instructions:

```
Mc <- "My car"
# Use the up arrow and change the last command to:
Yc <- "Your car"
# Save a file cars.Rhistory in the folder Cars.
savehistory("/path/to/Cars/cars.Rhistory")
q("no") # Exit R.
# Open R again.
# Note that the up arrow does not
# give access to the last two commands.
# Open the file cars.Rhistory in the folder Cars.
loadhistory(file.choose())
# Now, the up arrow can be used
# to find old commands.
q("no")
```

SECTION 9.4**Saving Plots**

You might also want to save plots you have produced with R. The instructions listed in this section have already been introduced in Chap. 7, but we give them again as a matter of interest. For example, to save the following plot:

```
> curve(cos(x), xlim=c(-10,10))
```



you simply need the command:

```
dev.print(png,file="myplot.png",width=480,height=480)
```

Another way of saving a plot is to first redirect the graphical device to a file, then execute the command to generate the plot.

```
png(file="myplot2.png",width=480,height=480)
curve(cos(x),xlim=c(-10,10))
dev.off()
```

Warning

Remember to use the function `dev.off()` at the end of the procedure: it closes the device and writes the plot to a file. Otherwise, the file will remain empty.



Other commands are available to save images in other formats: `jpeg()`, `png()`, `bitmap()`, `postscript()`, `pdf()`, ...

Under Microsoft Windows, you can also use the menu **File/Save as...** or copy-paste the plot to another software. You must first click on the graphics window to make it active.

Mac

You can use the menus **File/save**. The saved/copied plot is in the PDF format. Note that graphical windows have a “history”. You can explore the history of generated plots with the keystroke combination **COMMAND + left and right keyboard arrows**.



SECTION 9.5

Managing Packages

A package is a collection of data and functions belonging to a same theme.

When you install R, some basic functionalities come out of the box. But you can extend the functionalities of R by adding libraries, also called packages. First, install the package on the computer's hard disk, then load (activate) it in the memory of R only when needed (see Appendix on page 531 for further details).

First, you can use the function `search()` which gives the list of databases (collections of R packages) attached to the system. The function `searchpaths()` returns the same list, but adds the path to the corresponding file.

Recall that the function `library()` returns the list of all packages available in the library C:/PROGRAM FILES/R/R-3.1.0/library.

Do it yourself



```
search() # Return the list of databases attached to the system.  
library() # Return the list of packages saved on the disk.
```

Install package R2HTML and type in the following instructions:

```
library() # Package R2HTML is present on the disk.  
search() # Package R2HTML is not loaded.  
require("R2HTML") # Activate package R2HTML.  
search() # Package R2HTML is now loaded.
```

Note

The instructions `require("package")` and `library("package")` have a similar behaviour. The function `require()` is designed for use inside other functions; it returns FALSE and gives a warning (rather than an error as `library()` does by default) if the package does not exist.

SECTION 9.6

Managing Access Paths to R Objects

In the previous section, we saw the use of the function `search()`, which lists and numbers all databases attached to the system. We also saw how to load a package with the function `require()`. A database can also be attached with the function

`attach()` and detached with the function `detach()`. These two functions will be put to use in the practicals at the end of this chapter.

Tip

Suppose you have created a `data.frame` (individuals×variables table) called `mydata`. Then `attach(mydata)` attaches the `data.frame` `mydata`, giving access to the variables of the `data.frame` `data` directly by typing their names in the console.



The following instructions will help you understand how the function `attach()` works.

```
# Start R.
attach(file.choose()) # Open the file cars.RData from the folder
                      # Cars.
ls() # x is not mentioned.
x   # This is strange: the contents of x are displayed,
    # yet ls() does not mention it.
rm(x)
Warning message:
In rm(x): variable "x" cannot be found.
x   # Yet x is there!
```

The command `ls(pos=n)`, where `n` is an integer, returns the list of objects in the database at the `n`th position in the list given by `search()`.

For example, `ls(pos=2)` returns the objects for the module in second place, `ls(pos=3)` for the module in third place and so on.

Note that position 1 is reserved. Thus `ls()` is equivalent to `ls(pos=1)` and gives the list of all objects in the current workspace.

```
search()
ls(pos=2) # List all objects from the database cars.RData.
```

Do it yourself



```
require("datasets") # Load several datasets.
warpbreaks
mydata <- warpbreaks
fix(mydata)      # Read the data and the variables names.
breaks          # Returns an error message: this object
                 # is not defined.
search()         # Returns the list of databases attached
                 # to the system.
searchpaths()    # Same list, with the complete path.
position <- match("package:datasets",search())
              # Get the position of datasets in the list
              # output by search().
ls(pos=position) # Gives the list of all objects in datasets.
data()           # Description of these datasets.
```

```

attach(mydata)
search()
searchpaths()
ls(pos=2)
breaks      # No error message.

```

We now have direct access to the columns of `mydata`: `breaks`, `wool` and `tension`.

SECTION 9.7

† Other Useful Commands

In this section, we introduce a few other interesting commands to manage your work:

- Under Microsoft Windows, the menu **File/Save to file...** can be used to save to a text file all the text displayed in the console (including error messages); by default, the file is called `lastsave.txt` and is created in the current directory. The size of the contents is limited by parameters which can be changed in the menu **Edit/Preferences....**
- The function `sink(file="myoutput.txt")` redirects all **R** output (which would usually be displayed in the console) to the file `myoutput.txt`. To stop this functionality, type `sink()` in the console.
- The menu **File/Source R code...** is used to transfer a sequence of **R** instructions from a file directly to the console. This command also checks the syntax of the **R** code in the file before transferring it. Equivalently, you can type `source(file.choose())` in the console.
- **R** includes many functions to manage files and directories on the hard disk:
`file.create()`, `file.exists()`, `file.remove()`, `file.rename()`,
`file.append()`, `file.copy()`, `file.symlink()`, `dir.create()`,
`Sys.chmod()`, `Sys.umask()`, `file.info()`, `file.access()`, `file.path()`,
`file.show()`, `list.files()`, `unlink()`, `basename()`, `path.expand()`.
For example, the command `list.files()` returns a vector of strings of characters of the names of files in the specified directory. The command `file.exists()` is used to find out whether a file exists in a given directory. We refer the reader to the help files for the details of all these functions.

SECTION 9.8

† Problems in Memory Management

In this section, we shall focus on memory management by the computer in general and by R in particular. We shall try to understand why messages occur such as

`cannot allocate a vector of size xxx`

We shall also see how to work with high-dimension vectors and matrices.

Before giving indications on memory management in R, we need to give a short explanation of the internal workings of a computer. Upon execution of an R program, the following internal components of the computer are used:

- The hard disk (which contains the code and data files)
- The processor (which performs the calculations; there are 32 bit and 64 bit processors)
- And the RAM (for random access memory), which holds temporarily the data which are used by the processor for the calculations

In what follows, we shall mostly focus on the RAM, though we will also mention the processor. The main interest of the RAM as opposed to the hard disk is that it can be accessed very quickly. In a computer, the processor accesses the instructions of the program to execute and the data necessary to execution from the RAM.

9.8.1 Organization of RAM

The RAM is organized as an ordered sequence of boxes, and each box can contain a binary digit: 0 or 1. The information contained in a box, the smallest quantity of information that can be contained, is called a *bit* (for *binary digit*). At this point, it is worth noting that information is actually often organized in blocks of 8 boxes. Another unit has thus been introduced: the byte, which is worth 8 bits.

Warning

1 byte = 8 bits.



Also note that each block is numbered; the number of a (8 boxes) block is called its *memory address*. A memory address is thus an identifier, which designates a specific zone of memory where data (or instructions to run) can be read and stored. This identifier is usually an integer, often expressed in hexadecimal notation (base $b = 16$, see Sect. 5.9) (Fig. 9.1).

9.8.2 Accessing the Memory

Note

A process running on a computer does not usually have a direct access to the RAM but rather to the so-called virtual memory. The memory addresses used by R are thus addresses in the virtual memory; the operating system then puts these addresses in correspondence with the actual RAM memory addresses. We do not distinguish between the two types of memory.

To access a given zone of memory, R uses (in a transparent way hidden from the user) what is called a *pointer* (a quantity which “points” to the desired memory zone). A pointer is a variable containing a memory address. At the address contained in a given pointer, we can find for example a data point. Note that each data point has a specific type, such as integer, double, etc. (see Chap. 3). Note also that an integer is coded on 4 bytes, a double on 8 bytes, a character on 1 byte, a logical on 4 bytes and a complex on 16 bytes, to cite the most common variable types. This is true on a 32 bit processor and on a 64 bit processor. Examine now the following R instructions:

```
> x <- 3L # create the value 3, of type integer,
> # or equivalently:
> x <- as.integer(3)
```

Given what we just explained, we can assume that at the same time, a memory slot of 32 successive blocks (4 bytes of 8 bits each) is reserved (or allocated) and a pointer is created containing the address of (the first of) these boxes. In fact, the pointer must contain not only the address of variable x but also its type to know over how many boxes the variable is stored. For this reason, the pointers are said to be “typed”. When a typed pointer is incremented (*i.e.* when we need to add one unit to the address it contains), it is not necessarily incremented by 1, but by the size of the pointed type.

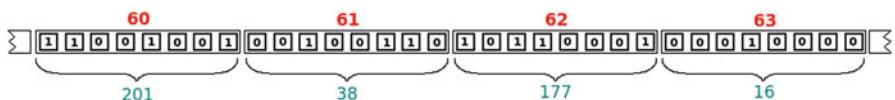


Fig. 9.1: Illustration of storage of values in memory. Each little box contains a binary number (0 or 1). Each green number gives the decimal representation of the number in binary form in the block above. Each red number gives the address (expressed here in decimal notation) of the 8-box block above. Note that the same memory addresses could have been written in hexadecimal notation ($b = 16$), giving 3C, 3D, 3E and 3F

9.8.2.1 Problems Caused by Memory Management of Integers

Since a (signed) integer is coded over 4 bytes, *i.e.* 32 bits, the largest integer that can be represented is 2,147,483,647. Indeed, if the first bit is reserved for the sign, there are 31 remaining available boxes, or 2^{31} possible arrangements. Counting 0, the largest available integer is thus $2^{31} - 1$, or

```
> as.integer(2^31-1)
[1] 2147483647
> .Machine$integer.max
[1] 2147483647
```

The result below is thus not surprising.

```
> as.integer(2^31)
[1] NA
```

The number 2^{31} can thus only be handled by R as a *double*:

```
> 2^31
[1] 2147483648
> is.double(2^31)
[1] TRUE
```

Warning

The largest vector that can be allocated in R is of length $2^{31} - 1 \sim 2 \cdot 10^9$, whether on a 32 bit or 64 bit processor. This is easy to understand: it corresponds to the largest integer that R can define, and the length on a vector (number of elements) is stored as a (signed) integer.



We note in passing that this knowledge on R's behaviour helps understand the output below.

```
> 46360*46360 # 46360 is stored as a double.
[1] 2149249600
> 46360L*46360L # 46360L is stored as an integer.
[1] NA
```

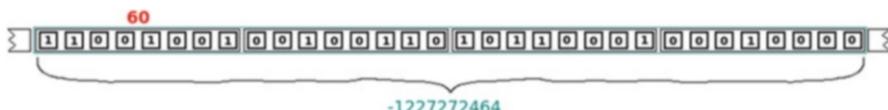


Fig. 9.2: Illustration of R storage in memory of a (signed) integer. Each little box contains a binary digit (0 or 1). The green number gives the decimal notation of the integer expressed in binary notation in the four blocks above. The red number gives the address (expressed here in decimal base) of the first 8-box memory block above. Note that here, a number is stored over 32 boxes and not over 8 as in Fig. 9.1. Furthermore, the first box is used to specify the sign of the number, negative here

```
> sum(1:304) # 1:304 is stored as an integer.
[1] 46360
> sum(1:304)*sum(1:304)
[1] NA
Warning message :
In sum(1:304) * sum(1:304) :
  NA produced by overflow
> 46360^2 # The result is stored as a double.
[1] 2149249600
> sum(1:304)^2 # The result is stored as a double.
[1] 2149249600
```

The *Warning message* above comes from the fact that `sum(1:304)` is an integer. Note that the exponent function (`^`) transforms its arguments into reals and returns a real number.

9.8.2.2 Successive Allocation of Memory

In fact, the smallest block of memory that can be allocated (reserved) by R is 8 bytes (=64 bits). Memory is thus allocated in R in blocks of 8 **successive** blocks (both on 32 and 64 bit processors). In the instruction `x <- 3L`, there are thus 64 reserved boxes (of 1 bit each), of which the first 32 are used to store the integer value +3. All 64 boxes would be used to allocate a double, with the instruction:

```
> x <- 3.0
```

Note



The 64 boxes are apportioned as follows: 1 box for the sign, 11 boxes for the exponent and 52 boxes for the significand, in the floating point representation (see Sect. 5.9.2).

The package associated with this book includes the functions `getaddr()` and `writeaddr()` which can be used respectively to get the memory address of a variable containing a number and to write a value at a memory address.

```
> x <- c(8L,9L)
> x
[1] 8 9
> addr <- getaddr(x)$addr.int # Gets the address of the first
                                # box of the 64-box block where x
                                # is stored.
> addr
[1] 53173920
> writeaddr(addr,6L) # Write the integer 6 at this address.
> x
[1] 6 9
> writeaddr(addr+4L,7L) # An integer is coded over 4 bytes,
                           # hence increment the address by 4 to
                           # get to x[2].
```

```
> x  
[1] 6 7
```

Now with a vector of doubles,

```
> x <- c(12.8,4.5)  
> x  
[1] 12.8 4.5  
> addr <- getaddr(x)$addr.int # Get the address of the first box  
# of the 128-box block where x is  
# stored.  
> writeaddr(addr,6.2)  
> x  
[1] 6.2 4.5  
> writeaddr(addr+8L,7.1) # A double is coded over 8 bytes.  
> x  
[1] 6.2 7.1
```

Advanced users

R can only access memory boxes that have been allocated by R, and other software cannot access the memory zones reserved by R. This is essential! Otherwise, the data for our calculations could be modified by external software. Even another R session cannot access the memory zones reserved by the first. For example, type in a first R session:

```
> x <- 1L  
> getaddr(x)$addr.int  
[1] 37602112
```

which is the address of the memory block containing the integer value 1. Then type in a second R session:

```
> writeaddr(37602112,7L)
```

Then (if the second R session does not crash!) we can check that the value of x in the first session has not been modified. Return to the first session and type

```
> x # The attempt to modify the value of x in another R session  
# is doomed to failure.  
[1] 1  
> # Try the modification in the same session:  
> writeaddr(37602112,7L)  
> # This time it works!  
> x  
[1] 7
```



9.8.3 Object Size in R

A useful R function gives the size of an object: the function `object.size()`. Given the previous subsection, we expect the instruction `object.size(3L)` to output 8 bytes, but that is not the case.

```
> object.size(3L) # (On a 64 bit processor.)
48 bytes
```

In fact, each R object contains (even when declaring a simple integer by `x <- integer(3)`) a header which takes up some space in the RAM: 24 bytes on a 32 bit processor and 40 bytes on a 64 bit processor. This header is used to save information of the created object: its type (*integer*, *double*, *complex*, etc.), its length, etc.

Tip

To find out on what kind of processor R is running, use the instruction:



```
> .Machine$sizeof.pointer
[1] 8
```

The value 8 is returned for a 64 bit processor and the value 4 for a 32 bit processor.

The values returned by the following instructions are rather clear now:

```
> # On a 32 bit processor:
> object.size(3L) - 24
8 bytes
> # On a 64 bit processor:
> object.size(3L) - 40
8 bytes
```

Advanced users



Memory allocation in R is done differently for small and large integer vectors. Small vectors belong to one of 6 classes, depending on their length (lesser than or equal to 2, 4, 8, 12, 16 or 32), and can store data of 8, 16, 32, 48, 64 or 128 bytes, respectively. Since an integer uses up only 4 bytes, these classes can be used to store, respectively, 2, 4, 8, 12, 16 and 32 integers. An integer vector of length $n > 32$ uses up space of size $4n + 40$ if n is even and $4(n + 1)$ if n is odd, to which we add a header of 2 bytes on a 32 bit processor and 40 bytes on a 64 bit processor. The following code returns the size in memory of vectors of increasing size.

```

> N <- 50
> V <- vector(length = 50)
> for (L in 1:N) {
+     z <- sample(N, L, replace = TRUE)
+     V[L] <- object.size(z)
+ }

> V # On a 32 bit processor.
[1]   8   8  16  16  32  32  32  32  48  48  48  48  48  64  64
[15]  64  64 128 128 128 128 128 128 128 128 128 128 128 128 128
[29] 128 128 128 128 136 136 144 144 152 152 152 160 160 160 168 168
[43] 176 176 184 184 192 192 200 200

> V # On a 64 bit processor.
[1]   8   8  16  16  32  32  32  32  48  48  48  48  48  64  64
[15]  64  64 128 128 128 128 128 128 128 128 128 128 128 128 128
[29] 128 128 128 128 136 136 144 144 152 152 152 160 160 160 168 168
[43] 176 176 184 184 192 192 200 200

```

9.8.4 Total Memory used by R

The total size of virtual memory allocated to R in a session includes:

- Memory used to store the values of objects (their contents)
- Memory used to store the headers of objects

This information can be accessed with the function `gc()`, in which `Ncells` represents the number of cells used for the header and `Vcells` the number of blocks for the values.

Do it yourself



The following example illustrates this function:

```
> rm(list=ls()) # Delete all objects in the session.
```

Type three times `gc()`. Note that the values displayed stabilize. Now type the instruction:

```
> x <- as.integer(3)
```

Type several times `gc()` until the results stabilize. Note that the value of `Vcells` has increased by 1 unit, corresponding to 8 bytes (smallest possible size of a block of data). Recall that an integer needs 4 bytes but is still stored in a memory zone of 8 bytes.

The total amount of memory available to R depends on several factors:

- The RAM physically present on the computer
- The RAM already used by the operating system and the other software being run on the system (such as a web browser if it is open)
- The type of processor (32 or 64 bit), since the RAM is limited to 4 GB (1 GB = 1024 kB) for 32 bit processors (and it is often closer to 3 GB or 2 GB), but is (really!) much larger for 64 bit processors

Note

 On a 32 bit processor, an address is coded over 32 bits, and hence the addressable memory is limited at 2^{32} bytes (=4 GB). On a 64 bit processor, an address is coded over 64 bits, so theoretically, the addressable memory is “limited” at 2^{64} bytes, a huge number. In fact, it is usually limited by the processor’s architecture. This information can be obtained from the manufacturer (it is called ‘Max Memory Size’).

Note also that R allocates memory for the creation of large objects and clears these objects from memory (when they are no longer used) by a process called *garbage collection*. You can force garbage collection with the function `gc()`.

Warning

 When creating a large object, the memory reserved by R must be contiguous (it cannot be fragmented in several blocks).

It is therefore possible that there remains enough total memory for R, but no “gap” large enough to fit the data of a single large object. Here is an illustration. Beware that these commands may provoke a major slowdown of your system or even a brutal crash of R.

```
> # First, type the instructions:
> # rm(list=ls()) ; gc() ; gc() # to empty the memory.
> P <- 14000
> D <- matrix(rep(0, P*P), nrow=P)
Error: cannot allocate a vector of size 1.5 GB.
> # But the next allocation is certainly possible,
> # after typing gc();gc() # to empty the memory.
> Q <- round(sqrt(P^2/2))
> D1 <- matrix(rep(0, Q*Q), nrow=Q)
> D2 <- matrix(rep(0, Q*Q), nrow=Q)
> # The sum of the sizes of D1 and D2 is approximatively 1.5 GB.
> object.size(D1) + object.size(D2)
1567843440 bytes
```

In the example above, it was not possible to create a single object of size 1.5 GB, but it was possible to create two objects of size 0.75 GB each.

Tip

Note that on a 64 bit processor, we would probably not have encountered this issue, even when creating an objects of size close to 3GB:

```
> # First we typed the instructions
> rm(list=ls()) ; gc() ; gc() # to empty the memory.
> P <- 20000
> D <- matrix(rep(0, P*P), nrow=P)
> object.size(D)
> 3200000200 bytes
```



9.8.5 A Few Recommendations

An elementary understanding of memory management on a computer in general, and in R in particular, will be very useful to help identify the origin of memory-related issues. Our first recommendation is thus to read the previous sections. We give here some extra recommendations.

For example, you could calculate the (approximative) size of a matrix before creating it. Since a real number uses up to 8 bytes, a real-valued matrix of size $n \times p$ needs $8np$ bytes. If you need to work with very large matrices, a 64 bit processor will allow you to allocate larger memory blocks. Compare this to a 32 bit processor which will usually not allow for more than 2 gigabytes. If you are not able to create a large object, remember to remove (using the function `rm()`) other useless large objects (the function `object.size()` gives the size of such objects) and to free up some memory with the function `gc()`. You can also close other software running on your computer to free up some memory and as a last resort purchase further physical memory.

Linux

The software `ksysguard` can be used to visualize in real time the memory used by R and by the other processes on your system.



Another option would be to split your matrix into several submatrices and find a way to perform your analysis on those, before combining the results.

Tip

The packages `bigmemory`, `ff` and `RevoScaleR` may be useful.



Under Windows, the functions `memory.size()` and `memory.limit()` display some information on the memory used. You can also read the online help: `help("Memory-limits")`.

Note

These problems, which stem from the conception of R, may be solved in the future. We refer the interested reader to the document <http://www.divms.uiowa.edu/~luke/talks/uiowall.pdf>.

SECTION 9.9**† Using R in BATCH Mode**

It is possible to launch a sequence of R instructions in BATCH mode. In this mode, R starts and automatically executes the instructions in the background and then closes down when the work is done.

- To start this mode, use the following instruction in a DOS window (or a terminal window under LINUX or Mac):

```
R CMD BATCH myfile.R myoutput.out
```

The file `myfile.R` should contain the list of R instructions to execute and the file `myoutput.out` will contain any messages and output displayed by R.

Warning

Note that you will have to set the PATH system variable to contain the path to the executable `Rgui.exe`. See the Warning frame on page [231](#).

- This mode is also useful when you want to start simulations on a remote UNIX/LINUX station (through a simple ssh tunnel). In this case, you should add the LINUX command `nohup`.

```
nohup /path/to/executable/R CMD BATCH myfile.R myoutput.out &
```

Linux

Under Linux, to find the `/path/to/executable/R`, you need to type in a terminal the instruction: `which R`.

It is also possible to create an R script that can be run without having to open R at first. To do this, download and modify to suit your needs the file <http://biostatisticien.eu/springeR/runthis.bat>.

See also

The interested reader can consult with profit the web page <http://cran.r-project.org/contrib/extra/batchfiles>.

**Linux**

Under Linux, create a script named `runthis` and make it executable (`chmod u+x runthis`). This script will contain the following lines:

```
#!/bin/bash
R --vanilla << "EOF" # Pipe all subsequent lines into R.

#####
# Put all your R code here #####
X11();plot(1:3);require("tcltk")
tkmessageBox(message="hello")
#####
# end of R code #####
EOF
```

**Tip**

If you want to pass command line arguments to R CMD BATCH, use the following approach. First, create a file called `test.R` containing the following lines:

```
args <- commandArgs(trailingOnly = FALSE)
print(args)
q("no")
```



Next, from the command line run:

R CMD BATCH -q -4 -foo test.R

Finally, issue:

cat test.Rout

— SECTION 9.10 —

† Creating a Simple R Package

A package is a practical way of grouping data sets, functions and help files in a single structure. This structure is stored in a `.zip` file (or `.tar.gz` under Linux). Nonetheless, this operation is complicated under Microsoft Windows, because it requires the installation of many tools which are not present by default on this operating system.

Mac

Specific documentation for Mac users will be made available on the website associated with this book.

You will need to install the following software:

- Latest version of Rtools available here: <http://cran.r-project.org/bin/windows/Rtools>.
- <http://www.biostatisticien.eu/springeR/htmlhelp.exe>.
- A complete version of Tex Live. Download the file <http://mirror.ctan.org/systems/texlive/tlnet/install-tl.zip> and unzip it in a temporary folder. Then double click on the file `install-tl.advance.bat`. A graphical installation interface pops open and will guide you through the installation of Tex Live.

Here is the procedure to create an R package:

- Start R.
- Import into the workspace the data sets and functions you want to include in your package.
- Use the function `package.skeleton()` to create the structure of your package. You should give a value to the following arguments:
 - `name`: a string of characters containing the name of the package
 - `list`: a vector of strings of characters, specifying the various objects (data sets and functions) to include in the package
 - `path`: a string of characters containing the path to the directory where your package structure will be created
- When you call this function, a folder is created in your current directory, containing the files and subfolders of your package. You then need to modify some of these files, as described in the file `Read-and-delete-me` which you will find in that folder.
- The last step is to create a `.zip` file containing the package structure. To this end, execute the following commands in an MS-DOS command window:
 - `R CMD check PackageName`
 - `R CMD build --binary --use-zip PackageName`

Tip

If your R code calls C/C++ or Fortran functions, the files containing the source code for these functions need to be placed in a subdirectory called `src/`, located in the folder whose name is specified by the argument `name` of the function `package.skeleton()`.

The practical section at the end of this chapter gives an example of package creation.

Advanced users

Linux users who do not have access to Microsoft Windows but wish to build a package for that operating system can use the website <http://win-builder.r-project.org/>, on which the Linux-created .tar.gz package can be uploaded. A Windows-compatible .zip package is then sent by e-mail to the address given in the Maintainer field of file DESCRIPTION.



Memorandum

`ls()`, `objects()`: list all objects available in the workspace
`rm()`: delete an object
`.RData`: extension for workspace files
`save.image()`: save all created objects in a file (`name.RData`)
`load()`: load a `.RData` file containing created objects
`.Rhistory`: extension for command history files
`savehistory()`: save the command history (`.Rhistory` file)
`loadhistory()`: load command history
`dev.print()`: save a plot
`search()`, `searchpaths()`: list of databases attached to the system
`attach()`: attach a database
`detach()`: detach a database
`require()`: load a package present on the disk
`sink()`: redirect R output to a .txt file
`source()`: import a sequence of R instructions from a file to the console
`package.skeleton()`: create a package structure



Exercises

- 9.1-** Name two R functions which return a list of objects in your session.
- 9.2-** How would you delete the object `foo`?
- 9.3-** Which R command gives the current directory?
- 9.4-** Which R command changes the current directory?
- 9.5-** What is the purpose of the function `save.image()`?
- 9.6-** What are the four things you can save before closing an R session?
- 9.7-** What is the purpose of the command history? Which keys are necessary to use it?
- 9.8-** What is the purpose of the function `history()`?
- 9.9-** Give the list of R instructions you would use to get a file called `myplot.png` containing a plot of the curve $y = x^2$.
- 9.10-** When is the function `attach()` useful for a `data.frame`?
- 9.11-** Which R function is used to load an R package to the memory?
- 9.12-** What is the purpose of the function `source()`?



Worksheet

Managing and Creating Packages

A- Using the Functions `attach()` and `detach()`

- 9.1- Download the file <http://www.biostatisticien.eu/springeR/bmichild.xls>.
- 9.2- Display the names of the variables of the data.frame.
- 9.3- Type GENDER. What do you observe?
- 9.4- Type `ls()`. Can you see the variable GENDER?
- 9.5- Use the function `attach()` on your data.frame, then type GENDER. What do you observe now?
- 9.6- Type `ls()` again. What do you observe?
- 9.7- Use the function `search()` to find the position at which your data.frame is attached.
- 9.8- Use the argument `pos` of the function `ls()` to list the objects present at this position.
- 9.9- Detach your data.frame and check (using the function `search()`) that it worked. Now type GENDER again and observe that this object has disappeared.
- 9.10- Create an object called GENDER containing the string "Male". Display the contents of this object.
- 9.11- Use the function `attach()` on your data.frame, then type GENDER. What do you observe?
- 9.12- Can you display the contents of the object GENDER of your data.frame? How about the object weight?
- 9.13- Type `ls()`. What do you observe? How about with `search()`?
- 9.14- Use the argument `pos` of the function `ls()` to check that the object GENDER of the data.frame does exist.
- 9.15- Use the function `get()` and its argument `pos` to display the contents of the object GENDER from your data.frame. Can you propose another approach?

B- Creating a mini-package

- Objects in the package

- 9.1- Start R, then change the current directory to the Windows Desktop, using the instruction `setwd(choose.dir())`.
- 9.2- Create the following functions and data sets:

```
f <- function(x,y) x+y
g <- function(x,y) x-y
d <- data.frame(a=1,b=2)
e <- rnorm(1000)
```

- **Package structure**

- 9.3-** Use the function `package.skeleton()` to create the structure of your package.

```
package.skeleton(name = "SmallRPkg", list = c("f", "g", "d", "e"))
```

A folder called `SmallRPkg` is created on your desktop. It contains three sub-folders (`data`, `man` and `R`) and two files (`DESCRIPTION` and `Read-and-delete-me`).

The folder `data` contains the files `d.RData` and `e.RData`, which contain, respectively, the data sets (in binary form) `d` and `e`, which you imported from the R console.

The folder `R` contains the files `f.R` and `g.R`, which contain the source code of the functions `f` and `g` defined earlier.

The folder `man` contains help files for all objects included in the package.

- 9.4-** You **must** edit the help files (.Rd extension files), even if they are not empty. Use the description of the help file for the function `mean` in Chap. 6. The fields to fill in are made apparent in all help files by lines starting with `%%`. Replace those lines (including the characters `%%`) with the appropriate information. Do not change the sentences starting with a single `%`. Furthermore, in fields of the form `keyword ~ kwd1`, you must replace `~ kwd1` with a reserved keyword; the list of reserved keywords is given by the instruction `file.show(file.path(R.home("doc"), "KEYWORDS"))`.

- 9.5-** You should also change the file `DESCRIPTION` and fill in the relevant fields. For example, it is **very important** that you give a valid e-mail address.

- 9.6-** You can then read and delete the file `Read-and-delete-me`.

Your package structure has now been created.

- **Creating the package file**

- 9.7-** You have one final operation to perform: building the `.zip` file which will include your structure (modified by R). You first need to change a few system environment variables. Use the key combination **WINDOWS+PAUSE** to open the system properties window, go to the section **System Variables** and edit the variable **PATH**. At the beginning of this long list of semi-colon-separated paths, add the path to the executable `Rgui.exe` and the path to the executable `hhc.exe` (**be careful not to delete anything!**).

- 9.8-** Open an MS-DOS command menu (using the menu **Start/Execute: command**) and execute the instructions

- `cd "C:\Documents and Settings\johndoe\Desktop"` (puts you in the folder containing the package structure).
- `R CMD check SmallRPkg`
Check that there are no error or warning messages here. If there are, make the suggested changes.
- `R CMD build --binary --use-zip SmallRPkg`

If there were no errors, the package file `SmallRPkg.zip` is created.

9.9- Install it from the following menu:

`Packages/Install package(s) from zip files...`

Read the help files of your package.

You can follow this procedure to create more complex packages, which you can then publicize.

Part III

Elementary Mathematics and Statistics

Chapter 10

Basic Mathematics: Matrix Operations, Integration and Optimization

Goals of this chapter

This chapter describes basic mathematical functions. It then gives some usual operations on matrices and the most usual decompositions. We also present a few numerical integration and differentiation functions and the main optimization functions.

SECTION 10.1

Basic Mathematical Functions

The following table is an almost exhaustive list of classical mathematical functions (Table 10.1).

Table 10.1: Table of basic mathematical functions

R name	Description	Example	Result
<code>x%%y</code>	Remainder of the division of x by y	<code>10%%3</code>	1
<code>ceiling()</code>	Smallest integer greater than or equal to x	<code>ceiling(2.3)</code>	3
<code>floor()</code>	Largest integer smaller than or equal to x	<code>floor(2.3)</code>	2
<code>round()</code>	Round the value of the first argument to the number of digits specified by the second argument	<code>round(2.375, 2)</code>	2.38
<code>signif()</code>	Round the value of the first argument to a given number of significant digits	<code>signif(2.375, 2)</code>	2.4
<code>trunc()</code>	Integer part of x , obtained by removing all digits after the decimal separator	<code>trunc(1.37)</code>	1
<code>sign()</code>	Sign ± 1	<code>sign(-2)</code>	-1
<code>abs()</code>	Absolute value $ x $	<code>abs(-2)</code>	2
<code>exp()</code>	Exponential e^x	<code>exp(0)</code>	1
<code>log()</code>	Natural logarithm	<code>log(1)</code>	0
<code>sqrt()</code>	Square root \sqrt{x}	<code>sqrt(4)</code>	2
<code>range()</code>	Range	<code>range(2,5,1)</code>	1 5
<code>max()</code>	Maximum	<code>max(2,3)</code>	3
<code>min()</code>	Minimum	<code>min(2,3)</code>	2
<code>sum()</code>	Sum of effective arguments	<code>sum(2,3,4)</code>	9
<code>prod()</code>	Product of effective arguments	<code>prod(2,4,2)</code>	16
<code>cummax()</code>	Cumulative maxima	<code>cummax(c(2,4,3))</code>	2 4 4
<code>cummin()</code>	Cumulative minima	<code>cummin(c(2,4,1))</code>	2 2 1
<code>cumsum()</code>	Cumulative sums	<code>cumsum(c(2,3,4))</code>	2 5 9
<code>cumprod()</code>	Cumulative products	<code>cumprod(c(2,4,3))</code>	2 8 24
<code>cos()</code>	Cosine	<code>cos(pi)</code>	-1
<code>sin()</code>	Sine	<code>sin(pi/2)</code>	1
<code>tan()</code>	Tangent	<code>tan(pi/4)</code>	1
<code>acos()</code>	Arccosine	<code>acos(1)</code>	0
<code>asin()</code>	Arsine	<code>asin(0)</code>	0
<code>atan()</code>	Arctangent	<code>atan(0)</code>	0
<code>cosh()</code>	Hyperbolic cosine	<code>cosh(0)</code>	1
<code>sinh()</code>	Hyperbolic sine	<code>sinh(0)</code>	0
<code>tanh()</code>	Hyperbolic tangent	<code>tanh(0)</code>	0
<code>acosh()</code>	Inverse hyperbolic cosine	<code>acosh(1)</code>	0
<code>asinh()</code>	Inverse hyperbolic sine	<code>asinh(0)</code>	0
<code>atanh()</code>	Inverse hyperbolic tangent	<code>atanh(0)</code>	0
<code>beta()</code>	Beta function	<code>beta(1,2)</code>	0.5
<code>lbeta()</code>	Logarithm of beta function $B(a, b)$	<code>lbeta(1,1)</code>	0
<code>factorial()</code>	Factorial $x!$	<code>factorial(6)</code>	720
<code>choose()</code>	Binomial coefficient $\binom{n}{p} = \frac{n!}{p!(n-p)!}$	<code>choose(5,2)</code>	10
<code>gamma()</code>	Gamma function $\Gamma(x)$ ($\Gamma(n) = (n-1)!$, if $n \in \mathbb{N}$)	<code>gamma(4)</code>	6
<code>lgamma()</code>	Logarithm of gamma function	<code>lgamma(2)</code>	0
<code>digamma()</code>	First derivative of gamma function	<code>digamma(2)</code>	0.4227843
<code>trigamma()</code>	Second derivative of gamma function	<code>trigamma(2)</code>	0.6449341

Note that most of these functions can take a vector as argument.

Do it yourself



- Check numerically for a few values that the following formula is correct:

$$\binom{n-1}{p-1} + \binom{n-1}{p} = \binom{n}{p}.$$
- Compute the sum of the first n integers for the values $n = 1, \dots, 10$. Check that this corresponds to the formula $\frac{n(n+1)}{2}$.
- Compute the sum of the squares of the first n integers for $n = 1, \dots, 10$. Check that this corresponds to the formula $\frac{n(n+1)(2n+1)}{6}$.
- For $x = (x_1, \dots, x_n)^T = (0.83, 0.13, -1.16, -1.14, -0.68, 0.73, -1.27)^T$, compute the value of

$$\hat{G}_n = \frac{n}{K} g_n(\widehat{m}_e, \hat{\theta})^2 \quad \text{where} \quad g_n(m_e, \theta) = 1 - \gamma - \frac{1}{n} \sum_{i=1}^n |y_i| \log |y_i|$$

with $y_i = \frac{x_i - m_e}{\theta}$, $\widehat{m}_e = \text{Median}(x_1, \dots, x_n)$, $\hat{\theta} = \frac{1}{n} \sum_{i=1}^n |x_i - \widehat{m}_e|$, $K = \frac{\pi^2}{3} - 3$, $\gamma = 1 - \psi(2)$ (Euler's constant) and $\psi(\cdot)$ the digamma function. Use the function `median()`.

Note: It is sufficient to check that the value \hat{G}_n is greater than the critical value `qchisq(1-alpha, df=1)`, for a given level α (usually 5 %), to decide, at error level α , that the data were not generated following a Laplace distribution. Such procedures are explored in detail in Chap. 13.

Tip

The number π can be used in R with the command `pi`.



SECTION 10.2

Matrix Operations

Several basic matrix operations are included in the base version of R. Before we introduce them, let λ be a scalar, let \mathbf{A} and \mathbf{B} be two real matrices and let \mathbf{C} be a complex matrix. We refer the interested reader to [29].

```
> lambda <- 2 # Creating scalar λ.
> A <- matrix(c(2,3,5,4), nrow=2, ncol=2) # Real matrix.
> A
     [,1] [,2]
[1,]    2    5
[2,]    3    4
> B <- matrix(c(1,2,2,7), nrow=2, ncol=2) # Symmetric real matrix.
> B
     [,1] [,2]
[1,]    1    2
[2,]    2    7
> C <- matrix(c(1,1i,-1i,3), ncol=2) # Hermitian complex matrix.
> C
     [,1] [,2]
[1,] 1+0i 0-1i
[2,] 0+1i 3+0i
> I2 <- diag(rep(1,2)) # Identity matrix of order 2.
```

We shall use these objects to illustrate matrix operations.

Note



For more sophisticated matrix manipulation, use the package **Matrix**.

10.2.1 Basic Matrix Operations

In R, the basic matrix operations are:

- Adding a scalar: $\lambda + \mathcal{A}$

```
> lambda+A
      [,1] [,2]
[1,]     4    7
[2,]     5    6
```

- Addition (entry-wise): $\mathcal{A} + \mathcal{B}$

```
> A+B
      [,1] [,2]
[1,]     3    7
[2,]     5   11
```

- Subtraction (entry-wise): $\mathcal{A} - \mathcal{B}$

```
> A-B
      [,1] [,2]
[1,]     1    3
[2,]     1   -3
```

- Multiplying by a scalar: $\lambda\mathcal{A}$

```
> lambda*A
      [,1] [,2]
[1,]     4   10
[2,]     6    8
```

- Transposition: \mathcal{A}^T

```
> t(A)
      [,1] [,2]
[1,]     2    3
[2,]     5    4
```

- Conjugation: $\bar{\mathcal{C}}$

```
> Conj(C)
 [,1] [,2]
[1,] 1+0i 0+1i
[2,] 0-1i 3+0i
```

- Entry-wise multiplication:

```
> A*B
 [,1] [,2]
[1,]    2    10
[2,]    6    28
```

- Dot product: $\mathcal{A}\mathcal{B}$

```
> A%*%B
 [,1] [,2]
[1,] 12 39
[2,] 11 34
```

- Entry-wise division:

```
> A/B
 [,1]      [,2]
[1,] 2.0 2.5000000
[2,] 1.5 0.5714286
```

- Matrix inversion: \mathcal{B}^{-1}

```
> solve(B)
 [,1]      [,2]
[1,] 2.3333333 -0.6666667
[2,] -0.6666667 0.3333333
```

- Matrix division: $\mathcal{A}^{-1}\mathcal{B}$

```
> solve(A)%%B # Identical to: solve(A,B)
 [,1]      [,2]
[1,] 0.8571429 3.857143
[2,] -0.1428571 -1.142857
```

- Cross product: $\mathcal{A}^T\mathcal{B}$

```
> crossprod(A,B) # t(A)%*%B
 [,1] [,2]
[1,]    8   25
[2,]   13   38
```

Do it yourself

Let $\mathcal{M}, \mathcal{N}, \mathcal{O}, \mathcal{P}$ be the following matrices:

$$\mathcal{M} = \begin{bmatrix} 1 & 1 \\ 2 & 3 \\ 4 & 1 \end{bmatrix}, \quad \mathcal{N} = \begin{bmatrix} 3 & 4 \\ 1 & 3 \\ 4 & 1 \end{bmatrix}, \quad \mathcal{O} = \begin{bmatrix} 3 & 4 & 2 \\ 1 & 3 & 2 \end{bmatrix} \text{ and } \mathcal{P} = \begin{bmatrix} 3 & 4 & 2 \\ 1 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Give the dimensions of the matrices $\mathcal{M}, \mathcal{N}, \mathcal{O}$ and \mathcal{P} . Calculate $\mathcal{M} + \mathcal{N}$, $\mathcal{M} - \mathcal{N}$, $3\mathcal{M}$, \mathcal{MO} , \mathcal{OM} , \mathcal{M}^T , \mathcal{P}^{-1} . Check that $\mathcal{PP}^{-1} = \mathcal{I}_3 = \mathcal{P}^{-1}\mathcal{P}$.

Let \mathcal{Q} and \mathcal{R} be the following matrices: $\mathcal{Q} = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$ and $\mathcal{R} = [3 \ 4 \ 1]$.

Calculate \mathcal{QR} , \mathcal{RQ} and $\mathcal{Q}^T \mathcal{P} \mathcal{Q}$.

10.2.2 Outer Product

The outer product of column vectors x and y is the matrix xy^T of general element $x_i y_j$.

```
> x <- seq(1,4)
> y <- seq(4,7)
> outer(x,y,FUN="*")
 [,1] [,2] [,3] [,4]
[1,]     4     5     6     7
[2,]     8    10    12    14
[3,]    12    15    18    21
[4,]    16    20    24    28
```

Tip

The function `outer()` allows more general operations than simple entry-wise multiplication. For example, the command `outer(x,y,FUN=f)` on vectors $x = (x_1, \dots, x_n)^T$, $y = (y_1, \dots, y_n)^T$ with the function $f(x, y)$ produces the following matrix:

$$\begin{pmatrix} f(x_1, y_1) & \cdots & f(x_1, y_n) \\ \vdots & f(x_i, y_j) & \vdots \\ f(x_n, y_1) & \cdots & f(x_n, y_n) \end{pmatrix}.$$



10.2.3 Kronecker Product

If \mathbf{A} is an $m \times n$ matrix and \mathbf{B} is a $p \times q$ matrix, then the Kronecker product of matrix \mathbf{A} by matrix \mathbf{B} is the matrix $\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}$ of dimensions $mp \times nq$.

```
> kronecker(A,B)
 [,1] [,2] [,3] [,4]
[1,]    2     4     5    10
[2,]    4    14    10    35
[3,]    3     6     4     8
[4,]    6    21     8    28
```

10.2.4 Triangular Matrices

It can be useful to get the lower and upper parts of a matrix. This can be done with the functions `lower.tri()` and `upper.tri()`.

```
> M <- matrix(1:16,nrow=4)
> lower.tri(M)
 [,1] [,2] [,3] [,4]
[1,] FALSE FALSE FALSE FALSE
[2,] TRUE FALSE FALSE FALSE
[3,] TRUE TRUE FALSE FALSE
[4,] TRUE TRUE TRUE FALSE
> upper.tri(M,diag=TRUE)
 [,1] [,2] [,3] [,4]
[1,] TRUE TRUE TRUE TRUE
[2,] FALSE TRUE TRUE TRUE
[3,] FALSE FALSE TRUE TRUE
[4,] FALSE FALSE FALSE TRUE
> M[lower.tri(M)] <- 0
> M
 [,1] [,2] [,3] [,4]
[1,]    1     5     9    13
[2,]    0     6    10    14
[3,]    0     0    11    15
[4,]    0     0     0    16
```

10.2.5 Operators *vec* and *Half vec*

The matrix operator *vec* applied to a matrix \mathcal{A} outputs the long column vector $vec(\mathcal{A})$ made by concatenating the columns of \mathcal{A} . It is computed in R with the following instruction:

```
> vec <- function(M) as.matrix(as.vector(M))
> # or equivalently, but outside a function call:
> # dim(A) <- c(prod(dim(A)),1)
> A
      [,1] [,2]
[1,]    2    5
[2,]    3    4
> vec(A)
      [,1]
[1,]    2
[2,]    3
[3,]    5
[4,]    4
```

The matrix operator *vech* (for *vec half*) applied to a matrix \mathcal{A} outputs the long column vector $vech(\mathcal{A})$ made by concatenating the columns of \mathcal{A} , but excluding elements above the diagonal of \mathcal{A} . It is computed in R with the following instruction:

```
> vech <- function(M) as.matrix(M[lower.tri(M,diag=TRUE)])
> vech(A)
      [,1]
[1,]    2
[2,]    3
[3,]    4
```

10.2.6 Determinant, Trace and Condition Number

The function `det()` computes the determinant of a matrix.

```
> det(A)
[1] -7
```

There is no R function to compute the trace of a matrix directly, but it is very easy to calculate:

```
> sum(diag(A))
[1] 6
```

Warning

 Do not use the function `trace()` to compute the trace of a matrix. This function is used to debug R code.

The condition number is the ratio of the largest and smallest non-zero singular values. A large condition number is an indicator that the matrix has bad numerical properties. It is computed with the function `kappa()`.

```
> kappa(A, exact=TRUE)
[1] 7.582401
```

10.2.7 Scaling and Centring Data

The function `scale()` is used to centre and/or scale a matrix. Centring corresponds to subtracting to each column the mean of that column. Scaling corresponds to dividing each column by its standard deviation.

Warning

Note that the function `sd()` calculates a standard deviation with $n - 1$ at the numerator.



Centring

```
> scale(A, scale=FALSE)
[,1] [,2]
[1,] -0.5  0.5
[2,]  0.5 -0.5
attr(,"scaled:center")
[1] 2.5 4.5
```

Scaling

```
> scale(A, center=FALSE, scale=apply
(A, 2, sd))
[,1]      [,2]
[1,] 2.828427 7.071068
[2,] 4.242641 5.656854
attr(,"scaled:scale")
[1] 0.7071068 0.7071068
```

Warning

To use a scaling factor based on the standard deviation of the population, as is done for example in the French school of data analysis, use the instruction:

```
> red <- sqrt((nrow(A)-1)/nrow(A))
> scale(A, center=FALSE, scale=apply(A, 2, sd)*red) # t(A/apply
(A, 2, sd))/red
[,1] [,2]
[1,] 4   10
[2,] 6   8
attr(,"scaled:scale")
[1] 0.5 0.5
```



10.2.8 Eigenvalues and Eigenvectors

The eigenvalues and eigenvectors of a matrix are returned by the function `eigen()`.

```
> eigen(A)
$values
[1] 7 -1
$vectors
[,1]      [,2]
[1,] -0.7071068 -0.8574929
[2,] -0.7071068  0.5144958
```

Tip

Note that if \mathcal{C} is a Hermitian matrix (*i.e.* a complex matrix equal to its own conjugate transpose), the function `eigen()` can be used to get the eigen-decomposition of \mathcal{C} , *i.e.* $\mathcal{C} = \mathcal{V}\mathcal{D}\mathcal{V}^*$ (where \mathcal{V}^* is the conjugate transpose of \mathcal{V}):



```
> C <- matrix(c(1,1i,-1i,3),ncol=2)
> e <- eigen(C,symmetric=TRUE)
> V <- e$vectors
> D <- diag(e$values)
> all.equal(C,V%*%D%*%t(Conj(V)))
[1] TRUE
```

10.2.9 Square Root of a Hermitian Positive-Definite Matrix

A square root of a positive-definite matrix \mathcal{C} is any matrix \mathcal{M} verifying $\mathcal{M}^*\mathcal{M} = \mathcal{C}$, where \mathcal{M}^* denotes the conjugate transpose (adjoint matrix) of \mathcal{M} . It is usually denoted as $\mathcal{C}^{1/2}$ even when it is not unique. If \mathcal{C} is Hermitian (*i.e.* a complex matrix equal to its own conjugate transpose or a symmetric real matrix), then $\mathcal{C}^{1/2}$ can be computed as follows:

```
> e <- eigen(C,symmetric=TRUE)
> V <- e$vectors
> V %*% diag(sqrt(e$values)) %*% t(Conj(V)) # C^{1/2},
# which is
# Hermitian in this
# case.
[,1]      [,2]
[1,] 0.9238795+0.0000000i 0.000000-0.3826834i
[2,] 0.0000000+0.3826834i 1.689246+0.0000000i
```

The matrix $\mathcal{C}^{-1/2}$ can be computed as follows:

```
> V %*% diag(1/sqrt(e$values)) %*% t(Conj(V)) # C^{-1/2},
# which is
# Hermitian in
# this case.
[,1]      [,2]
[1,] 1.194478+0.0000000i 0.0000000+0.2705981i
[2,] 0.000000-0.2705981i 0.6532815+0.0000000i
```

10.2.10 Singular Value Decomposition

We wish to write $\mathbf{C} = \mathbf{U}\mathbf{D}\mathbf{V}^*$ where \mathbf{D} denotes the diagonal matrix of the singular values of \mathbf{C} and \mathbf{U} (respectively \mathbf{V}) denotes the matrix of left (respectively right) singular vectors of \mathbf{C} . To this end, we use the function `svd()`.

```
> res <- svd(C)
> res
$d
[1] 3.4142136 0.5857864
$u
[,1] [,2]
[1,] -0.3826834+0.0000000i 0.9238795+0.0000000i
[2,] 0.0000000-0.9238795i 0.0000000-0.3826834i
$v
[,1] [,2]
[1,] -0.3826834+0.0000000i 0.9238795+0.0000000i
[2,] 0.0000000-0.9238795i 0.0000000-0.3826834i
> D <- diag(res$d)
> U <- res$u
> V <- res$v
> all.equal(C,U%*%D%*%t(Conj(V))) # A = UDV*.
[1] TRUE
```

Tip

To compute the Moore–Penrose pseudo-inverse of a (non-invertible) matrix, use the following function:

```
> mpinv <- function(M,eps=1e-13) {
  s <- svd(M)
  e <- s$d
  e[e>eps] <- 1/e[e>eps]
  return(s$v%*%diag(e)%*%t(s$u))
}
```



10.2.11 Cholesky Decomposition

Let \mathbf{B} be a symmetric and positive-definite real matrix. We wish to write $\mathbf{B} = \mathbf{U}^\top \mathbf{U} = \mathcal{L} \mathcal{L}^\top$, where \mathbf{U} (respectively \mathcal{L}) is an upper (respectively lower) triangular matrix. Note that this implies that \mathbf{U} is a square root of \mathbf{B} . To get this decomposition, use the function `chol()`.

```
> U <- chol(B) # This is another method for calculating
# B^{1/2}.
> L <- t(U)
> U
```

```

[,1]      [,2]
[1,]    1 2.000000
[2,]    0 1.732051
> all.equal(B,t(U).%*%U) #  $\mathcal{B} = \mathcal{U}^T \mathcal{U}$ .
[1] TRUE

```

Note that the function `chol2inv()` can be used to compute the inverse \mathcal{B}^{-1} of a square symmetric and positive definite matrix \mathcal{B} , using its Cholesky decomposition.

```

> B
[,1] [,2]
[1,] 1 2
[2,] 2 7
> chol2inv(U) # This is  $\mathcal{B}^{-1}$ .
[,1]      [,2]
[1,] 2.3333333 -0.6666667
[2,] -0.6666667  0.3333333
> all.equal(chol2inv(U), solve(B))
[1] TRUE

```

Finally, note that the function `chol()` can be used to compute $\mathcal{B}^{-1/2}$ as follows:

```

> solve(chol(B))# This is a version of  $\mathcal{B}^{-1/2}$ .
[,1]      [,2]
[1,] 1 -1.1547005
[2,] 0 0.5773503

```

10.2.12 QR Decomposition

We wish to write $\mathcal{A} = \mathcal{Q}\mathcal{R}$, where \mathcal{Q} is an orthogonal matrix ($\mathcal{Q}\mathcal{Q}^T = \mathcal{Q}^T\mathcal{Q} = \mathcal{I}$) and \mathcal{R} is an upper triangular matrix.

```

> res <- qr(A)
> Q <- qr.Q(res)
> Q
[,1]      [,2]
[1,] -0.5547002 -0.8320503
[2,] -0.8320503  0.5547002
> all.equal(I2,Q.%*%t(Q)) #  $\mathcal{I}_2 = \mathcal{Q}\mathcal{Q}^T$ .
[1] TRUE
> R <- qr.R(res)
> R
[,1]      [,2]
[1,] -3.605551 -6.101702
[2,] 0.000000 -1.941451
> all.equal(A,Q.%*%R) #  $\mathcal{A} = \mathcal{Q}\mathcal{R}$ .
[1] TRUE

```

Tip

Note that `qr(A, tol=1e-07)$rank` returns the rank of the matrix `A` using `tol` as the tolerance for detecting linear dependencies in the columns of `A`.



— SECTION 10.3 —

Numerical Integration

The numerical value of an integral can be computed in `R` with the function `integrate()`.

A few examples will help illustrate this function. Suppose you wish to check numerically that

$$\int_{-\infty}^{\infty} \frac{\exp(-x^2/2)}{\sqrt{2\pi}} dx = 1.$$

Use the following procedure:

```
> myf <- function(x) {exp(-x^2/2)/sqrt(2*pi)}
> integrate(myf,lower=-Inf,upper=Inf)$value
[1] 1
```

Note that it is also possible to integrate functions of several variables. Thus, suppose you wish to check numerically that

$$\int_{x=0}^{x=1} \int_{y=2}^{y=3} \cos(x+y) dy dx = 2\cos(3) - \cos(4) - \cos(2).$$

Use the following procedure:

```
> myf <- function(x) {
+   res <- vector("integer",length(x))
+   for (i in 1:length(x)) {
+     res[i] <- integrate(f=function(y,x){cos(x+y)},lower=2,
+                           upper=3,x=x[i])$value
+   }
+   return(res)
+ }
> integrate(myf,lower=0,upper=1)$value
[1] -0.9101945
> 2*cos(3)-cos(4)-cos(2)
[1] -0.9101945
```

Note

Note that the function `integrate()` also returns the precision of the numerical calculation.



Do it yourself

Check numerically that the function $f_X(x) = \frac{1}{2} \left(1 + \frac{(x-1)}{8}\right)^{-5} \mathbb{1}_{[1,+\infty]}(x)$ is a probability distribution function, *i.e.* that it integrates to 1. It is in fact the density function of a generalized Pareto distribution with parameters (1,2,1/4).

— SECTION 10.4 —

Differentiation**10.4.1 Symbolic Differentiation**

R has very limited capabilities for symbolic calculus and we shall not dwell on this aspect. Note however that R includes symbolic differentiation functions: D() and deriv(). For instance,

```
> D(expression(sin(cos(x + y^2))), "x")      # Differentiate with
                                                # respect to
                                                # x.
- (cos(cos(x + y^2)) * sin(x + y^2))
> D(expression(sin(cos(x + y^2))), "y")      # Differentiate with
                                                # respect to
                                                # y.
- (cos(cos(x + y^2)) * (sin(x + y^2) * (2 * y)))
> f <- deriv(~ x^2, "x", TRUE)                # Differentiate
                                                # x^2
                                                # and get
                                                # 2x.

> f(3)  # Returns 3^2 and 2*3
[1] 9
attr(", "gradient")
  x
[1,] 6
```

Advanced users

The package Ryacas interfaces R with the formal calculus software Yacas, available online at <http://yacas.sourceforge.net>. Install and load the package Ryacas, then try the command vignette("Ryacas").



10.4.2 Numerical Differentiation

Numerical differentiation can be done in R with the function `grad()` from the package `numDeriv`.

```
> require("numDeriv")
> f <- function(x) x^2 # Function of one variable.
> grad(f,c(2,1,3,5)) # Computes the derivative at several
# (scalar) points.
[1] 4 2 6 10
```

This package also includes the functions `hessian()` to compute second-order derivatives but only at a single (vectorial) point.

```
> g <- function(x) x[1]*x[2]^2 # Function of two variables.
> grad(g,c(2,1)) # Calculates the derivative at a single
# (vectorial) point.
[1] 1 4
> hessian(g,c(2,1)) # Calculates the second-order derivative
# at a single (vectorial) point.
[,1] [,2]
[1,] 4.210428e-14 2
[2,] 2.000000e+00 4
```

The function `numericDeriv()`, which is trickier to use, returns the gradient of a function of several variables at several vectorial points. For instance, the following instructions compute the gradient vector of the function xy^2 at (2, 1) and (3, 4). The results are (1, 4) and (16, 24).

```
> h <- function(x,y) x*y^2 # Function of two variables.
> x <- c(2,3)
> y <- c(1,4)
> attributes(numericDeriv(quote(h(x,y)),c("x","y")))$gradient
[,1] [,2] [,3] [,4]
[1,] 1 0 4 0
[2,] 0 16 0 24
```

— SECTION 10.5 —

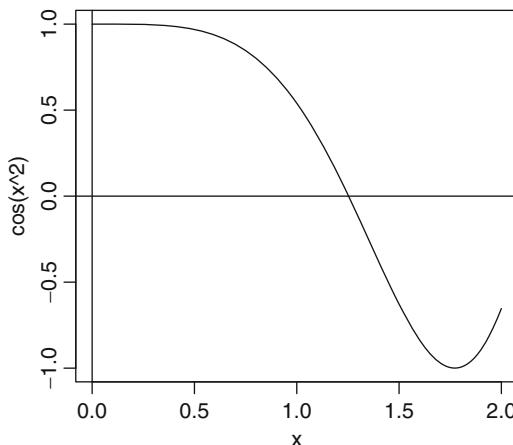
Optimization

10.5.1 Optimization Functions

Optimizing a function means finding where it reaches its maximum or its minimum. There are several R functions to this end, based on various algorithms: `optimize()`, `optim()`, `nlm()`, `constrOptim()` and `nlminb()`.

- **One-dimensional optimization**

The function `optimize()`, which only works in one dimension, is the easiest to manipulate. As an example, suppose you wish to find the value and argument of the minimum of the function $\cos(x^2)$ over the interval $[0, 2]$. This function is plotted on the next figure.



The function `optimize()` solves this problem numerically.

```
> optimize(f=function (x) {cos (x^2)}, lower=0, upper=2)
$minimum
[1] 1.772453
$objective
[1] -1
```

The minimum is -1 and is reached at $x = 1.772453$.

Tip

The argument `maximum=TRUE` of the function `optimize()` is used to compute the maximum rather than the minimum of a function.

- **Multidimensional optimization**

We wish to find the maximum of the function of two variables $f(x, y) = 10 \frac{\sin(\sqrt{(x-3)^2 + (y-4)^2})}{((x-3)^2 + (y-4)^2)^{\alpha/2}}$ with $\alpha = 1.1$. We use the function `nlm()`, which calculates the minimum of a function. The function f is plotted on Fig. 10.1 using the following R instructions:

```
> x <- seq(-10,10,length= 30)
> y <- x
> f <- function(x,y,alpha=1.1) { r <- sqrt((x-3)^2+(y-4)^2)
+                                10 * sin(r)/r^alpha }
> z <- outer(x, y, f)
> z[is.na(z)] <- 1
> op <- par(bg = "white")
> persp(x,y,z,theta=30,phi=30,expand=0.5,
+        col="lightblue",ticktype="detailed")
```

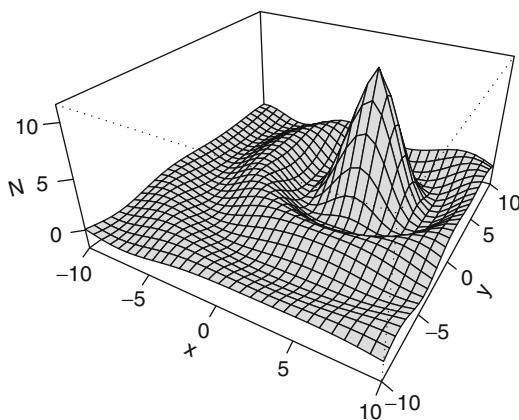


Fig. 10.1: Modified sinc function

Finding the maximum of f is equivalent to finding the minimum of $-f$, so we use `nlm()` on the function $-f$.

```
> f <- function(z,alpha=1.1) {
+   x <- z[1]
+   y <- z[2]
+   r <- sqrt((x-3)^2+(y-4)^2)
+   return(-10 * sin(r)/r^alpha)
+ }
> res <- nlm(f,c(0,0)) # The second effective argument gives
# the initial values.
> res
$minimum
[1] -1.046464
$estimate
[1] -1.627385 -2.169848
$gradient
[1] -1.534979e-07  1.139977e-07
$code
[1] 1
$iterations
[1] 7
```

The maximum is $(-1) \times \text{res\$minimum} = 1.046464$ and is reached at $\text{res\$estimate} = (-1.627, -2.169)$.

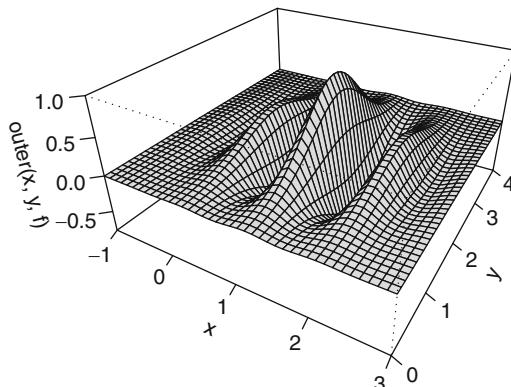
Tip

In `nlm()`, you can specify the value of a parameter of the function to minimize. Thus, to maximize f with $\alpha = 2$, use the following instruction:

- **Constrained optimization**

For optimization subject to simple constraints, such as $-a \leq x \leq b$ and $-c \leq y \leq d$, use the function `nlminb()` and its arguments `lower` and `upper`. For example, suppose we wish to find the three maxima of the surface of equation $e^{-(x-1.2)^2-(y-2)^2} \cos(2\pi(x-1.2))$ over the range $[-1, 3] \times [0, 4]$, whose plot is obtained using the following instructions:

```
> f <- function(x,y) exp(-(x-1.2)^2-(y-2)^2)*cos((x-1.2)*pi*2)
> x <- seq(-1,3,0.1)
> y <- seq(0,4,0.1)
> persp(x,y,outer(x,y,f),theta=30,phi=30,expand=0.5,
+           col="lightblue",ticktype="detailed")
```



First, we search for the global maximum. Visual inspection shows that it is reached in the range $[0.5, 1.5] \times [0, 4]$.

```
> f <- function(x) -exp(-(x[1]-1.2)^2-(x[2]-2)^2)*
+                           cos(2*pi*(x[1]-1.2))
> nlminb(c(0.8,0),f,lower=c(0.5,0),upper=c(1.5,4))
$par
[1] 1.200000 2.000000
$objective
[1] -1
```

```
$convergence
[1] 0
$message
[1] "relative convergence (4)"
$iterations
[1] 19
$evaluations
function gradient
 27        42
```

The (global) maximum is $(-1) \times \text{res\$objective} = 1$ and is reached at $\text{res\$par} = (1.2, 2)$.

We conclude with a search over the ranges $[-1, 0.5] \times [0, 4]$ and $[1.5, 3] \times [0, 4]$, with the following instructions:

```
nlminb(c(0,0),f,lower=c(-1,0),upper=c(0.5,4))
nlminb(c(2,0),f,lower=c(1.5,0),upper=c(3,4))
```

Note

For linear constraints such as

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \geq 0 \Leftrightarrow \begin{cases} ax + by - c_1 \geq 0 \\ cx + dy - c_2 \geq 0 \end{cases}$$

you can use the function `constrOptim()` with arguments `ui`= $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and `ci`= $\begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$.



10.5.2 Roots of a Function

The roots (or zeros) of a function f are defined as the solutions to the equation $f(x) = 0$.

- **Single root**

If there is a single root, use the function `uniroot()` which performs a one-dimensional search over a given interval. For example, here is how you would find the value at which the function $\cos(x^2)$ vanishes, over the interval $[0, 2]$ (we mentioned this function when we introduced one-dimensional optimization). The analytical solution to this simple problem is $x = \sqrt{\pi/2} = 1.253314$.

```
> uniroot(f=function(x){cos(x^2)},lower=0,upper=2,
+           tol=0.00001)$root
[1] 1.253314
```

- **Roots of a polynomial**

The function `polyroot()` finds all (possibly complex) roots of a polynomial. For example, this instruction finds the roots of the polynomial $P(x) = 3 - 8x + x^2$.

```
> polyroot(c(3,-8,1)) # These are (8+c(-1,1)*sqrt(54))/2.  
[1] 0.3944487+0i 7.6055513+0i
```

Memorandum

`round()`: round numbers
`abs()`: absolute value
`sqrt()`: square root
`exp()`, `log()`: exponential, logarithm
`max()`, `min()`: maximum, minimum
`sum()`, `prod()`: sum, product
`cummax()`, `cummin()`, `cumprod()`, `cumsum()`: cumulative maxima, minima, products and sums
`cos()`, `sin()`: cosine, sine
`%*%`: matrix multiplication operator
`t()`, `solve()`: matrix transposition, inversion
`outer()`, `kronecker()`: external product, Kronecker product
`det()`: matrix determinant
`eigen()`: eigenvalues and eigenvectors of a matrix
`svd()`, `chol()`, `qr()`: matrix decompositions (singular values, Cholesky, QR)
`integrate()`: numerical integration
`D()`, `deriv()`: symbolic differentiation
`grad()`, `hessian()`: numerical differentiation with package `numDeriv`
`optimize()`, `nlm()`, `nlmnb()`: function optimization
`uniroot()`, `polyroot()`: roots of a function



Exercises

- 10.1-** Which function calculates binomial coefficients?
- 10.2-** Give the instruction to compute the sum of the first n integers.
- 10.3-** Which function returns the range of a sample?
- 10.4-** What is the output of this instruction:
`matrix(c(1,0,0,1),nrow=2)*matrix(1:4,nrow=2)`?
- 10.5-** What is the symbol for matrix multiplication?
- 10.6-** Which function transposes a matrix? Which function inverses a matrix?
- 10.7-** Give the instruction to create the identity matrix of size 5.
- 10.8-** Give the instructions to calculate the determinant and the trace of a matrix.
- 10.9-** Give the instruction to centre and scale a matrix \mathcal{A} .
- 10.10-** Which function calculates the eigenvalues and eigenvectors of a matrix?
- 10.11-** Give the instruction to integrate numerically the function $3x^2 + 2$ over $[-1, 1]$.
- 10.12-** Give the instruction to find the maximum of the function $\sin^2(x)$ over $[0, 2]$.
- 10.13-** Which function would you use to find where a function vanishes? Where a polynomial vanishes?



Worksheet

Matrix Operations, Optimization and Integration

A- A First Optimization Problem

The aim of this practical is to find the eigenvalues of the following matrix, using several methods:

> **A**

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

- 10.1-** Create a function called `myf()` which evaluates the characteristic polynomial $P(x) = \det(\mathbf{A} - x\mathbf{I}_2)$ at point x (hint: use the function `det()`). Recall that the eigenvalues of a matrix are the roots of its characteristic polynomial.
- 10.2-** Change the function `myf()` so that it takes vector values.
- 10.3-** Plot the function `myf()` over the range $[-10, 10]$. Add axes.
- 10.4-** Use the function `uniroot()` twice to find the two roots of this function.
- 10.5-** Find the coefficients of the polynomial $P(x)$, then use the function `polyroot()` to calculate the roots of this polynomial.
- 10.6-** Check your results with the function `eigen()`.

B- A Second Optimization Problem

The following information is given about the figure below:

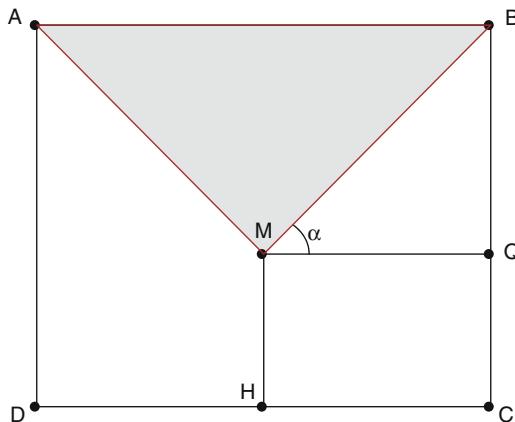
- (MH) is the perpendicular bisector of $[DC]$.
- Q is the orthogonal projection of M onto (BC) .
- $g(\alpha) = MA + MB + MH$ with $\alpha = \widehat{AMB} \in]0; \pi/2]$.
- $AB = 10$ and $BC = 6$.

The aim of this practical is to find the angle α which minimizes $g(\alpha)$.

Recall the following trigonometric identities:

- $\cos(\theta) = \frac{\text{length of adjacent side}}{\text{length of hypotenuse}}$
- $\sin(\theta) = \frac{\text{length of opposite side}}{\text{length of hypotenuse}}$
- $\tan(\theta) = \frac{\text{length of opposite side}}{\text{length of adjacent side}} = \frac{\sin(\theta)}{\cos(\theta)}$
- $\sin(\pi/2 - \theta) = \cos(\theta)$

- 10.1-** Reproduce the figure with R.
- 10.2-** Show analytically that $g(\alpha) = 5(2 - \sin(\alpha))/\cos(\alpha) + 6$.
- 10.3-** In R, create the function `g`.
- 10.4-** Calculate numerically the value and argument of the minimum of $g(\alpha)$.
- 10.5-** Calculate analytically $g'(\alpha)$.



- 10.6-** Check your result with symbolic differentiation.
- 10.7-** In R, create this function, which you can call `gprime`.
- 10.8-** Calculate numerically the root of $g'(\alpha)$. Check that you get the same result as before.

C- Standard Normal Table

We shall use the function `integrate()` to create a table for the distribution $\mathcal{N}(0, 1)$.

Let $\Phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt$ be the cumulative density function of $\mathcal{N}(0, 1)$. It is well known that $\Phi(-x) = 1 - \Phi(x)$. We shall thus only create the table for positive values of x .

- 10.1-** Create a function `phi()` which takes as input a vector x of length n and returns the vector of values $\frac{1}{\sqrt{2\pi}} e^{-x_i^2/2}$, $i = 1, \dots, n$.
- 10.2-** Use the function `integrate()` to compute $\Phi(x)$ for all x in the following vector: `quantiles <- seq(0, 5.5, by=0.1)`. Store these values in a vector called `probs`.
- 10.3-** Use the function `all.equal()` to compare these results with those given by the function `pnorm()`.
- 10.4-** Plot the values of $\Phi(x)$ for all x and all $-x$ in `quantiles` (hint: use the function `rev()`).
- 10.5-** Plot the function `pnorm()` in blue over the previous curve. Check that the two plots coincide perfectly.

D- Principal Components Analysis

Principal components analysis is used to describe the proximity of individuals on which several quantitative traits have been measured. This method requires many

matrix operations and is thus a good way to put into practice the notions introduced at the beginning of this chapter.

Data were collected in the wine region of Bordeaux. They include:

- Four weather traits:
 - TEMPER: sum of daily average temperatures (in degrees Celsius)
 - SUN: length of insolation (in hours)
 - HEAT: number of days of strong heat
 - RAIN: rainfall (in mm)
- Wine quality (QUALITY), as determined by wine-tasters:
1 = good wine, 2 = average wine, and 3 = mediocre wine

We shall represent these quantitative data on a scatter plot after projecting the data onto a subspace (a plane) chosen so as to limit the loss of information caused by the projection.

- 10.1-** Import into the variable `climatewine` the contents of the data file <http://www.biostatisticien.eu/springeR/climatewine.csv>
- 10.2-** Store in matrix `X` the variables TEMPER, SUN, HEAT, RAIN (hint: `as.matrix()`).
- 10.3-** Calculate the centre (of gravity) `g` of the scatter plot of individuals included in `X` (vector of column averages) with the function `colMeans()`. Display it with two decimal places.
- 10.4-** Use the function `scale()` to calculate the matrix $\dot{\mathcal{X}} = (\dot{x}_{ij})$ of centred data, and store it in the variable `Xdot`.
- 10.5-** Calculate the global *inertia* of the scatter plot, which represents the dispersion of the points: $\mathcal{I} = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^p \dot{x}_{ij}^2$, where n is the number of individuals in \mathcal{X} .
- 10.6-** The contribution of individual i to the global inertia is given by the formula $\frac{1}{n\mathcal{I}} \sum_{j=1}^p \dot{x}_{ij}^2$, where p is the number of variables (four in this case). Calculate the vector `inertiacontr` of contributions to inertia from each individual.
- 10.7-** Create the column matrix `onen` of length n , containing only 1s.
- 10.8-** Check that the centre of gravity `g` of the scatter plot is also given by the formula $\mathbf{g} = \frac{1}{n} \mathcal{X}^\top \mathbf{1}_n$.
- 10.9-** Check that the matrix `Xdot` of centred data is also given by the formula $\dot{\mathcal{X}} = \mathcal{X} - \mathbf{1}_n \mathbf{g}^\top$.
- 10.10-** Calculate the matrix of covariances `S` with the formula $\mathcal{S} = \frac{1}{n} \dot{\mathcal{X}}^\top \dot{\mathcal{X}}$. Try to reproduce this result with the function `cov()`.
- 10.11-** Use the matrix `S` to calculate the diagonal matrix `Doneovers` containing the inverses of standard deviations: $\mathcal{D}_{1/s} = diag(1/s_1, \dots, 1/s_p)$.
- 10.12-** Calculate the matrix `Z` of scaled and centred data with the formula $\mathcal{Z} = \dot{\mathcal{X}} \mathcal{D}_{1/s}$.

- 10.13-** Calculate the matrix of correlations R with the formula $\mathcal{R} = \frac{1}{n} \mathcal{Z}^T \mathcal{Z}$. Try to reproduce this result with the function `cor()`.
- 10.14-** Calculate the diagonal matrix Λ of eigenvalues (Λ) and the matrix W of eigenvectors (W) of the correlation matrix \mathcal{R} . The matrix W contains the coordinates of a new basis in which we shall represent the individuals.
- 10.15-** Plot the circle of correlations, which is a circle of radius 1, over which you should overlay arrows starting at the origin and ending at the points with coordinates given by the first two columns of the matrix $W\Lambda^{1/2}$. On the same plot, add the names of the variables at the end of the four arrows.
- 10.16-** The total inertia of the scaled and centred scatter plot is equal to the number of variables (four in this case). It can be decomposed into a sum of inertias contributed by each one of the four axes of the basis of W , which are given by the diagonal of Λ . Calculate the vector of percentages of total inertia explained by the first k axes ($1 \leq k \leq p$). What percentage of the inertia is explained by the first two axes?
- 10.17-** Calculate the matrix C_W of principal components with the formula $C_W = ZW$. The principal components are the coordinates of the individuals in the new basis described by W .
- 10.18-** Open a new graphics window and plot the individuals projected onto the first principal plane, *i.e.* with coordinates given by the first two columns of C_W . Add the names of the individuals onto the plot.
- 10.19-** Draw the last plot again, but this time plot in red (respectively blue and green) good wines (respectively average wines and mediocre wines). Add a caption.
- 10.20-** The quality of representation of individual i on the first principal plane is given by the formula $\frac{c_{i1}^2 + c_{i2}^2}{\sum_{j=1}^p c_{ij}^2}$, where c_{ij} is the entry at row i and column j of the matrix C_W . Calculate the vector QLT of qualities of representation of individuals on the first principal plane. Display QLT with two decimal places.
- 10.21-** We shall briefly explore package `ade4` which performs PCA. Install and load this package.
- 10.22-** Type the following instructions:

```
rownames(X) <- climatewine[,1]
res <- dudi.pca(X) # Answer 2 to the question you are asked.
scatter(res)
s.class(res$li,as.factor(climatewine[,6]))
```

Chapter 11

Descriptive Statistics

Goals

This chapter describes the procedures in R to structure your variables, draw standard summary plots of your data and calculate simple numerical statistical summaries on a data set. The data used to illustrate this chapter are from the data set NUTRIELDERLY. We also give a few examples of functions to produce prettier plots, useful for presentations or reports.

— SECTION 11.1 —

Introduction

All examples in this chapter are based on the data file `nutrition_elderly.xls` which you can import into R using one of the methods described in Chap. 4. For example, you could use the following instructions (remember to install the file <http://www.biostatisticien.eu/springeR/Rtools.exe> if you are working under Microsoft Windows):

```
> require("gdata") # Gives access to the function read.xls().  
> nutrielderly <- read.xls(  
+ "http://www.biostatisticien.eu/springeR/nutrition_elderly.xls")  
  
> attach(nutrielderly)  
> head(nutrielderly)  
   gender situation tea coffee height weight age meat fish  
1      2          1    0      0    151     58    72    4    3  
2      2          1    1      1    162     60    68    5    2  
3      2          1    0      4    162     75    78    3    1  
4      2          1    0      0    154     45    91    0    4  
5      2          1    2      1    154     50    65    5    3  
6      2          1    2      0    159     66    82    4    2
```

	<code>raw_fruit</code>	<code>cooked_fruit_veg</code>	<code>chocol</code>	<code>fat</code>
1	1	4	5	6
2	5	5	1	4
3	5	2	5	4
4	4	0	3	2
5	5	5	3	2
6	5	5	1	3

This table includes measures of 13 variables for 226 individuals.

In the remainder of the chapter, we shall assume that the data used for the various examples have been structured as explained in the next section.

— SECTION 11.2 —

Structuring Variables According to Type

The 13 variables from the data set NUTRIELDERLY can be sorted by type with the algorithm given in the next figure. Recall that the type of a variable X is determined by the set E_X of **observable** modalities and not by the modalities which are actually observed (Fig. 11.1).

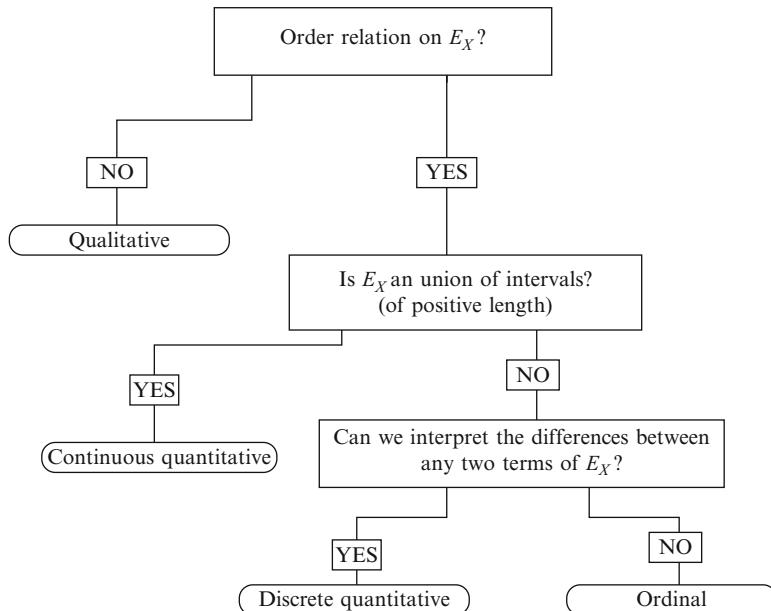


Fig. 11.1: Algorithm to determine the type of a variable

With this algorithm, we get the following summary table:

Qualitative variables	gender, situation and fat
Ordinal variables	meat, fish, raw.fruit, cooked.fruit.veg and chocol
Discrete quantitative variables	tea and coffee
Continuous quantitative variables	height, weight and age

First of all, we shall set up an adapted R structure for each variable.

11.2.1 Structuring Qualitative Variables

For qualitative variables, set up the structure with the function `as.factor()`. It might be useful to also use the function `levels()` to recode the modalities of a qualitative variable.

Let us perform these operations on the qualitative variables from our data set:

```
> gender <- as.factor(gender)
> levels(gender) <- c("Male", "Female")
> situation <- as.factor(situation)
> levels(situation) <- c("single", "couple", "family", "other")
> fat <- as.factor(fat)
> levels(fat) <- c("butter", "margarine", "peanut",
+                   "sunflower", "olive", "Isio4", "rapeseed", "duck")
```

Note that if the variable is coded as presence/absence, you can also use an R structure with a vector of logical values:

```
> smoker <- c(1,0,0,1,0,1,0,1,0,0) # 10
                                # smokers (=1) / non-smokers (=0).
> smoker
[1] 1 0 0 1 0 1 0 1 0 0
> smoker <- as.logical(smoker)
> smoker
[1] TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
[10] FALSE
```

If you also wish to associate a name to your modalities, you can proceed as follows:

```
> smoker <- c(smokes=smoker)
> smoker
smokes1 smokes2 smokes3 smokes4 smokes5 smokes6 smokes7
  TRUE    FALSE    FALSE     TRUE    FALSE    TRUE    FALSE
smokes8 smokes9 smokes10
  TRUE    FALSE    FALSE
```

However, we advise against coding presence/absence variables as logical values, as this hinders their use in other R functions, especially for graphical representation.

Advanced users

If you start with a variable which is already structured into factors, for example,

```
> smoker <- as.factor(c("Smoker", "NonSmoker", "NonSmoker",
+                         "Smoker", "NonSmoker", "Smoker", "NonSmoker",
+                         "Smoker", "NonSmoker", "NonSmoker"))
> smoker
[1] Smoker      NonSmoker   NonSmoker   Smoker      NonSmoker
[6] Smoker      NonSmoker   Smoker      NonSmoker   NonSmoker
Levels: NonSmoker Smoker
```



you can code it as logical values with the following procedure:

```
> smoker <- c(smoke=as.logical(2-as.integer(smoker)))
> smoker
smoke1 smoke2 smoke3 smoke4 smoke5 smoke6 smoke7
FALSE    TRUE     TRUE    FALSE    TRUE    FALSE    TRUE
smoke8 smoke9 smoke10
FALSE    TRUE     TRUE
```

11.2.2 Structuring Ordinal Variables

For ordinal variables, the structure can be set up with the function `as.ordered()`. It can also be useful to recode the modalities of an ordinal variable with the function `levels()`.

Let us perform these operations on the ordinal variables from our data set:

```
> meat <- as.ordered(meat)
> fish <- as.ordered(fish)
> raw_fruit <- as.ordered(raw_fruit)
> cooked_fruit_veg <- as.ordered(cooked_fruit_veg)
> chocol <- as.ordered(chocol)
> mylevels <- c("never", "< 1/week.", "1/week.", "2-3/week.",
+               "4-6/week.", "1/day")
> levels(chocol) <- levels(cooked_fruit_veg) <-
+                     levels(raw_fruit) <- mylevels
> levels(fish) <- levels(meat) <- mylevels
```

11.2.3 Structuring Discrete Quantitative Data

For a discrete variable, the structure is set up with the function `as.integer()`.

```
> tea <- as.integer(tea)
> coffee <- as.integer(coffee)
```

This only works if the data take integer values.

11.2.4 Structuring Continuous Quantitative Variables

For a continuous variable, the structure is set up with the function `as.double()`.

```
> height <- as.double(height)
> weight <- as.double(weight)
> age <- as.double(age)
```

— SECTION 11.3 —

Data Tables

For a given data set, plots and numerical summaries which can be performed depend strongly on the structure of the data table and on the type of the variable(s) in play. In this chapter, we detail how data can be organized into tables in R.

11.3.1 Individual Data Tables

This is the most common organization type. The data consist of measures of one or several variables for each of N individuals in a population. Refer to Chap. 4 to find out how to import data into R. These data are usually organized in a `data.frame`.

11.3.2 Tables of Counts and Frequency Tables

It is often interesting to represent a table of individual data (or raw data table) in a more condensed form. A table of counts or a table of frequencies makes it easier to understand the distribution of a variable, especially if the data are qualitative or ordinal. Such tables are obtained with the function `table()`.

```
> tc <- table(fat)           # Table of counts.
> tc
fat
  butter margarine    peanut sunflower      olive      Isio4
      15       27        48         68        40        23
  rapeseed      duck
      1         4
> tf <- tc/length(fat)    # Frequency table.
> tf
fat
  butter   margarine     peanut   sunflower      olive
0.066371681 0.119469027 0.212389381 0.300884956 0.176991150
  Isio4   rapeseed      duck
0.101769912 0.004424779 0.017699115
```

```
> levels(fat)                      # Levels.
[1] "butter"      "margarine"   "peanut"     "sunflower"  "olive"
[6] "Isio4"        "rapeseed"    "duck"       #
> nlevels(fat)                   # Number of levels.
[1] 8
```

11.3.3 Tables of Grouped Data

It is sometimes interesting to represent a table of individual data (or raw data table), of one or several quantitative variables, in a more condensed form. To this end, use a table of grouped data, giving the count (or frequency) of predefined classes.

Note that you can use the function `hist()` and specify the boundaries of classes in the argument `breaks`; it will then return the count in each of these classes. The default value of the argument `breaks` is "Sturges", which automatically calculates the classes.

```
> res <- hist(height,plot=FALSE)
> nn <- as.character(res$breaks)
> x <- as.table(res$counts)
> dimnames(x) <- list(paste(nn[-length(nn)],nn[-1],sep="-"))
> x
 140-145 145-150 150-155 155-160 160-165 165-170 170-175
      1       7      37      50      46      31      27
 175-180 180-185 185-190
      17       4       6
> # Or using the function cut():
> table(cut(height,res$breaks,include.lowest=TRUE))
 [140,145] (145,150] (150,155] (155,160] (160,165] (165,170]
      1       7      37      50      46      31
 (170,175] (175,180] (180,185] (185,190]
      27      17       4       6
```

11.3.4 Cross Tabulation

11.3.4.1 Contingency Tables

When given a table of individual data, you can use the function `table()` to get the observed contingency table for a couple (X, Y).

```
> mytable <- table(gender,situation)
> mytable
      situation
gender   single couple family other
  Male      20     63      2      0
Female     78     56      7      0
```

To add margins to this table, use the function `addmargins()`.

```
> table.complete <- addmargins(mytable, FUN=sum, quiet=TRUE)
> table.complete
      situation
gender   single couple family other sum
Male       20     63      2     0  85
Female     78     56      7     0 141
sum       98    119      9     0 226
```

Tip

- To change the label (“sum”) of the margins in the previous table, you can define the function `Total()` (thus: `Total <- sum`) and then replace `sum` with `Total` in the above call of `addmargins()`.
- If the contingency table is already given in a file, see Chap. 4 on how to import such a file with the function `read.ftable()` and display it with the function `ftable()`.



11.3.4.2 Joint Distribution

The joint distribution table (or table of relative frequencies) for a couple (X, Y) is obtained from the contingency table `mytable`.

```
> freqtable <- mytable/sum(mytable)
> freqtable
      situation
gender   single     couple     family     other
Male     0.088495575 0.278761062 0.008849558 0.000000000
Female   0.345132743 0.247787611 0.030973451 0.000000000
```

To get margins, use one of the following instructions:

```
> Total <- sum
> addmargins(freqtable, FUN=Total, quiet=TRUE)
      situation
gender   single     couple     family     other
Male     0.088495575 0.278761062 0.008849558 0.000000000
Female   0.345132743 0.247787611 0.030973451 0.000000000
      Total    0.433628319 0.526548673 0.039823009 0.000000000
      situation
gender      Total
Male     0.376106195
Female   0.623893805
Total    1.000000000

> freqtabletotal <- table.complete/table.complete[
+                           length(table.complete)]
> freqtabletotal
```

```

            situation
gender      single     couple    family     other
Male   0.088495575 0.278761062 0.008849558 0.000000000
Female 0.345132743 0.247787611 0.030973451 0.000000000
sum    0.433628319 0.526548673 0.039823009 0.000000000
            situation
gender      sum
Male   0.376106195
Female 0.623893805
sum    1.000000000

```

11.3.4.3 Marginal Distributions

The marginals of a distribution or contingency table `freqtable` are obtained with the function `margin.table()`.

```

> margin.table(freqtable,1) # Right margin.
gender
  Male   Female
0.3761062 0.6238938
> margin.table(freqtable,2) # Bottom margin.
situation
  single     couple    family     other
0.43362832 0.52654867 0.03982301 0.00000000

```

Warning

When the distribution (or contingency) table is complete, i.e. when it already includes the margins (as is the case for `freqtabletotal`), do not extract them with the function `margin.table()`, but proceed thus:



```

> freqtabletotal[,ncol(freqtabletotal)] # Right margin.
  Male   Female      sum
0.3761062 0.6238938 1.0000000
> freqtabletotal[nrow(freqtabletotal),] # Bottom margin.
  single     couple    family     other      sum
0.43362832 0.52654867 0.03982301 0.00000000 1.00000000

```

11.3.4.4 Conditional Distributions

Conditional distribution tables are obtained with the function `prop.table()`.

- Conditional distribution of `situation` knowing the values of `gender` :

```

> prop.table(mytable,1)
            situation
gender      single     couple    family     other
Male     0.23529412 0.74117647 0.02352941 0.00000000

```

```

Female 0.55319149 0.39716312 0.04964539 0.00000000
> addmargins(prop.table(mytable,1),margin=2,FUN=sum) # Add Total.
      situation
gender      single     couple    family     other
Male       0.23529412 0.74117647 0.02352941 0.00000000
Female    0.55319149 0.39716312 0.04964539 0.00000000
      situation
gender      sum
Male      1.00000000
Female   1.00000000

```

- Conditional distribution of gender knowing situation :

```

> prop.table(mytable,2)
      situation
gender      single     couple    family other
Male       0.2040816 0.5294118 0.2222222
Female    0.7959184 0.4705882 0.7777778
> addmargins(prop.table(mytable,2),margin=1,FUN=sum) # Add Total.
      situation
gender      single     couple    family other
Male       0.2040816 0.5294118 0.2222222
Female    0.7959184 0.4705882 0.7777778
sum      1.0000000 1.0000000 1.0000000

```

SECTION 11.4 –

Numerical Summaries

For simplicity, we present all numerical summaries on the vector $\mathbf{x} = (x_1, \dots, x_N)^T$. This vector is the set of N values of the variable X measured on a population of size N (standard situation in descriptive statistics). When the vector \mathbf{x} represents a subsample, we shall denote its length by n . Examples are mainly based on the data vector `height`.

Warning

Numerical summaries cannot be computed when some data are missing (`NA`). If necessary, missing data can be omitted with the function `na.omit()`.

```
> x <- na.omit(height) # Useless in this case since height has
# no NA.
```



11.4.1 Summaries of the Location of a Distribution

11.4.1.1 Modes

The modes of variable X are the values that occur most frequently. They can be computed for a variable of any type, although for continuous quantitative variables, the modal class should be used instead. Note that the mode may be unique, in which case the distribution is said to be unimodal, as opposed to multimodal.

```
> tabtea <- table(tea)
> names(which.max(tabtea))           # Getting a unique mode.
[1] "0"
> names(tabtea)[max(tabtea) == tabtea] # Getting all modes.
[1] "0"
```

In this case, the variable `tea` (number of cups of tea per day) is unimodal.

Tip

For a quantitative variable, you can use the function `as.numeric()` on the above results to get numerical values.

11.4.1.2 Median

The median of a statistical series is the value m_e which splits the series into two halves (larger and smaller than m_e) of same size, when the values are sorted from smallest to largest. It is a location criterion, which is obviously relevant only for purely qualitative variables. To compute it, there are two cases:

- If the total sample size N of the series is odd, then the median is the value at position $\frac{N+1}{2}$.
- If the total sample size N is even, then any value between the values at positions $\frac{N}{2}$ and $\frac{N}{2} + 1$ can be used as a median of the series. In practice, the median is usually the mean between these two values (this method thus excludes ordinal characters).

The R function to calculate the median, **only for numerical data**, is `median()`.

```
> median(x)
[1] 163
```

Tip

We propose the following code, which computes the median for **individual** ordinal or numerical data:

```
> my.median <- function(x) {
+   if (is.numeric(x)) return(median(x))
+   if (is.ordered(x)) {
+     N <- length(x)
+     if (N%%2) return(sort(x)[(N+1)/2]) else {
+       inf <- sort(x)[N/2]
+       sup <- sort(x)[N/2+1]
+       if (inf==sup) return(inf) else return(c(inf,sup))
+     }
+   }
+   stop("Cannot calculate the median for this type of data.")
+ }
> my.median(fish)
[1] 2-3/week.
6 Levels: never << 1/week. < 1/week. < ... < 1/day
```



Now suppose that we do not have individual data, but only the table of observed frequencies $f_1, \dots, f_k, \dots, f_K$ in the K classes $[e_0, e_1], \dots, [e_{k-1}, e_k], \dots, [e_{K-1}, e_K]$ of a **quantitative** variable. In that case, we need a linear interpolation between the two values with cumulative frequencies closest to 50 %. Let e_k and e_{k+1} be the two consecutive values of class boundaries such that $CFP_X(e_k) < 0.5 \leq CFP_X(e_{k+1})$, where $CFP_X(e_j) = f_1 + f_2 + \dots + f_j$, $j = 1, \dots, K$ ($CFP_X(x)$ is the value of the cumulative frequency polygon at point x). Then,

$$m_e = e_k + (e_{k+1} - e_k) \frac{0.5 - CFP_X(e_k)}{CFP_X(e_{k+1}) - CFP_X(e_k)}.$$

We propose the following code to compute the median in such a case:

```
> median.for.freq <- function(x) { # x is the frequency table.
  tmp <- suppressWarnings(as.double(names(x)))
  if (!is.null(tmp) | all(is.na(tmp))) {
    tab.freq.cum <- cumsum(x)
    index <- order(tab.freq.cum < 0.5) [1]
    fc1 <- tab.freq.cum[index]
    fc2 <- tab.freq.cum[index-1]
    x1 <- as.numeric(names(fc1))
    x2 <- as.numeric(names(fc2))
    mex <- as.numeric(x1 + (x2-x1)*(0.5-fc1)/(fc2-fc1))
  } else {mex <- NA}
  return(mex)
}
```

Here is how we compute the median for the data in vector **x** grouped into classes with the function **hist()**:

```
> res <- hist(x, plot=FALSE, breaks=c(130, 150, 160, 170, 180, 190))
> tab.x <- table(rep(res$breaks[-1], res$counts))
```

```
> tab.x
150 160 170 180 190
 8   87  77  44  10
> median.freq(tab.x/sum(tab.x))
[1] 162.3377
```

11.4.1.3 Mean

It can only be computed for quantitative variables.

```
> mean(x)      #  $\mu_X = \frac{1}{N} \sum_{i=1}^n x_i$ .
[1] 163.9602
```

11.4.1.4 Quantiles

The quantile of order p ($0 < p < 1$) is the value q_p of the variable X which splits the sample into two parts, one of size equal to p % of the total sample size of x (elements smaller than q_p), the other of size $(1 - p)$ % (elements greater than q_p). It cannot be computed for purely qualitative variables.

Quantiles of order 10 % and 90 %:

```
> quantile(x,probs=c(0.1,0.9))
10% 90%
153 176
```

Quartiles $q_{1/4}, q_{1/2}, q_{3/4}$ (also denoted as $q_1, q_2 = m_e, q_3$):

```
> quantile(x,probs=c(0.25,0.5,0.75))
25% 50% 75%
157 163 170
```

Deciles:

```
> quantile(x,probs=1:10/10)
 10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
153.0 156.0 158.5 160.0 163.0 165.0 168.0 172.0 176.0 188.0
```

Tip

The function `summary()` applied to a vector of quantitative data calculates the minimum, maximum, mean and the three quartiles.

11.4.2 Summaries of the Dispersion of a Distribution

These summaries can only be computed for quantitative variables. We first define the following three R functions and then present the summaries in the next table.

```
> # Variance  $\sigma^2$  of the population  $\sigma^2(x) = \frac{1}{N} \sum_{i=1}^N (x_i - \mu_X)^2$ .
> var.pop <- function(x) var(x)*(length(x)-1)/length(x)
> # Standard deviation  $\sigma$  of the population
>  $\sigma(x) = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu_X)^2}$ .
> sd.pop <- function(x) sqrt(var.pop(x))
> # Coefficient of variation:
> #  $c_v = \frac{\sigma}{\mu_X}$ .
> co.var <- function(x) sd.pop(x)/mean(x)
```

Name	Mathematical formula	R instruction	Result
Range	$\max_{1 \leq i \leq N} (x_i) - \min_{1 \leq i \leq N} (x_i)$	max(x)-min(x) diff(range(x))	48.0
Interquartile range	$q_{3/4} - q_{1/4}$	IQR(x)	13.0
Variance	$\sigma_{Pop}^2(x) = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$	var.pop(x)	80.702
Standard deviation	$\sigma_{Pop}(x) = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2}$	sd.pop(x)	8.983
Coefficient of variation	$c_v = \frac{\sigma_{Pop}}{\mu_X}$	co.var(x)	0.055
Absolute deviation to the median	$\frac{1}{N} \sum_{i=1}^N x_i - me_X $	mad(x)	10.378
Mean absolute deviation	$\frac{1}{N} \sum_{i=1}^N x_i - \bar{x} $	mean(abs(x-mean(x)))	7.312

Note

An unbiased estimator $\hat{\sigma}^2$ of the variance σ^2 of a population, based on a sample of size n , can be computed with the function `var()`. The corresponding standard deviation $\hat{\sigma}$ is computed with the function `sd()`.

```
> var(x)      #  $\hat{\sigma}^2(x) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$ 
# with  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ .
[1] 81.06063
> sd(x)       #  $\hat{\sigma}(x) = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$ 
# with  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ .
[1] 9.003368
```



11.4.3 Summaries of the Shape of a Distribution

These summaries can only be computed for quantitative variables. It is worth mentioning coefficients of asymmetry (skewness) and of peakedness (kurtosis); the relevant R code is given below.

```

> skew <- function(x) mean((x-mean(x))^3)/sd.pop(x)^3
> skew(x)    #  $\gamma_1 = \frac{\mu_3}{\sigma^3}$ 
               # with  $\mu_3 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu_X)^3$ .
[1] 0.4256203
> kurt <- function(x) mean((x-mean(x))^4)/sd.pop(x)^4
> kurt(x)   #  $\beta_2 = \frac{\mu_4}{\sigma^4}$ 
               # with  $\mu_4 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu_X)^4$ .
[1] 2.778185

```

Tip

The functions `skewness()` and `kurtosis()` from package `moments` perform the same operations.

— SECTION 11.5 —

Measures of Association

11.5.1 Measures of Association Between Two Qualitative Variables

11.5.1.1 Pearson's χ^2 Statistic

The results of this section are obtained with the function `chisq.test()`. We present how to compute the contingency table of observed counts O_{ij} , the contingency table of theoretical (expected) counts E_{ij} ($1 \leq i \leq p, 1 \leq j \leq q$) (also called independence table) and the table of contributions to the χ^2 as well as the χ^2 itself.

```

> genderfat <- table(gender,fat)      # Observed contingency table
                                         # of  $O_{ij}$ 's.
> tab.ind <- chisq.test(genderfat)$expected  # The  $E_{ij}$ 's.
> round(tab.ind)
  fat
gender  butter margarine peanut sunflower olive Isio4
  Male      6       10      18       26      15      9
  Female     9       17      30       42      25     14
  fat
gender  rapeseed duck
  Male      0       2
  Female     1       2
> # (genderfat-tab.ind)^2/tab.ind:  $\frac{(O_{ij}-E_{ij})^2}{E_{ij}}$ .
> tab.contr <- chisq.test(genderfat)$residuals^2
> tab.contr

```

```

fat
gender      butter   margarine      peanut   sunflower
Male     3.367083116 0.002361810 0.233489502 0.818473834
Female    2.029801879 0.001423786 0.140756083 0.493406212
fat
gender      olive    Isio4    rapeseed      duck
Male     1.632483082 1.54046803 0.376106195 1.486777720
Female    0.984121007 0.928650954 0.226730685 0.896284441
> chi2 <- chisq.test(genderfat)$statistic # sum(tab.contr)
> chi2          #  $\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$ .
X-squared
15.15842

```

Tip

Another way to compute Pearson's χ^2 statistic is to use the function `summary()`.

```

> chi2 <- summary(table(gender,fat))$statistic
> chi2
[1] 15.15842

```



11.5.1.2 Φ^2 , Cramér's V and Pearson's Contingency Coefficient

All indicators in the next table are computed from the χ^2 coefficient.

```

> N <- sum(genderfat)
> p <- nrow(genderfat)
> q <- ncol(genderfat)

```

Indicator	Formula	Instruction R	Result
Φ^2	$\Phi^2 = \frac{\chi^2}{N}$	Phi2 <- chi2/N	0.067
Cramér's V^2	$V^2 = \frac{\Phi^2}{\min(p-1,q-1)}$	Phi2/((min(p,q)-1)	0.067
Pearson's contingency coefficient	$C = \sqrt{\frac{\chi^2}{(N+\chi^2)}}$ $= \sqrt{\frac{\Phi^2}{(1+\Phi^2)}}$	sqrt(chi2/(N+chi2))	0.251

Tip

The function `cramer.v()` from package `rgrs` can also be used to compute Cramér's V .

```

> require("rgrs")
> cramer.v(genderfat)^2
[1] 0.06707265

```



11.5.2 Measures of Association Between Ordinal Variables (or Ranks)

11.5.2.1 Kendall's τ and τ_b

This coefficient is based on the notion of concordance of individuals. For two individuals i and j and for ordinal variables X and Y having p and q modalities, respectively, pairs (x_i, y_i) and (x_j, y_j) are said to be concordant if $\text{sign}(x_j - x_i) = \text{sign}(y_j - y_i)$ and discordant if $\text{sign}(x_j - x_i) = -\text{sign}(y_j - y_i)$. If $x_i = x_j$ or $y_i = y_j$ (or both), the corresponding pair is neither concordant nor discordant: we say there is a tie. If there are n_c concordant pairs, n_d discordant pairs and n_t ties, then $n_c + n_d + n_t = \frac{1}{2}N(N - 1)$. Kendall's τ_b is then given by the formula

$$\tau_b = \frac{2(n_c - n_d)}{\sqrt{(N^2 - \sum_{k=1}^p n_{k\bullet}^2)(N^2 - \sum_{l=1}^q n_{\bullet l}^2)}}$$

where $2(n_c - n_d) = \sum_{k=1}^p \sum_{l=1}^q \text{sign}(x_k - x_l)\text{sign}(y_k - y_l)$, $n_{k\bullet} = \sum_{l=1}^q n_{kl}$ and $n_{\bullet l} = \sum_{k=1}^p n_{kl}$, with n_{kl} being the number of individuals having the modalities k for X and l for Y .

When there are no ties, this formula simplifies to what is known as Kendall's τ :

$$\tau = \frac{2(n_c - n_d)}{N(N - 1)}.$$

These two quantities can be computed in R with the function `cor()`.

```
> cor(as.numeric(meat), as.numeric(fish), method="kendall")
[1] -0.1583088
```

Tip

Note that you could also compute these coefficients yourself by translating the formula for τ_b into R instructions:



```
> Kendall.taub <- function(x,y) {
+   a1 <- sign(outer(as.numeric(x),as.numeric(x),""))
+   a2 <- sign(outer(as.numeric(y),as.numeric(y),""))
+   num <- sum(a1*a2)
+   N <- length(x)
+   b1 <- sum(margin.table(table(x,y),1)^2)
+   b2 <- sum(margin.table(table(x,y),2)^2)
+   denom <- sqrt((N^2-b1)*(N^2-b2))
+   taub <- num / denom
+   return(taub)
+ }
> Kendall.taub(meat,fish)
[1] -0.1583088
```

11.5.2.2 Spearman's Rank Correlation Coefficient ρ

First compute the ranks (function `rank()`) x_i and y_i of all individuals for variables X and Y . In case of a tie, assign to tied values the rank equal to the mean of their shared positions (which is what the function `rank()` does by default).

When there are no ties, Spearman's rank correlation coefficient is given by the formula

$$\rho = 1 - \frac{6 \sum_{i=1}^N d_i^2}{N(N^2 - 1)} \quad \text{with } d_i = x_i - y_i.$$

When there are ties, we need to use the Pearson's standard correlation coefficient between ranks:

$$\rho = \frac{N(\sum_{i=1}^N x_i y_i) - (\sum_{i=1}^N x_i)(\sum_{i=1}^N y_i)}{\sqrt{N(\sum_{i=1}^N x_i^2) - (\sum_{i=1}^N x_i)^2} \sqrt{N(\sum_{i=1}^N y_i^2) - (\sum_{i=1}^N y_i)^2}}.$$

To compute Spearman's rank correlation coefficient in R, you can use the functions `rank()` and `cor()` or use directly the function `cor()` with the argument `method` set to "spearman":

```
> cor(rank(fat), rank(situation))
[1] 0.008787643
> cor(as.numeric(fat), as.numeric(situation), method="spearman")
[1] 0.008787643
```

11.5.3 Measures of Association Between Two Quantitative Variables

11.5.3.1 Covariance and Pearson's Correlation Coefficient

For two quantitative variables, the appropriate measure of association is correlation. It is defined as the ratio of the covariance of the two variables and their respective standard deviations. It is computed with the function `cor()`.

```
> cor(height, weight)
[1] 0.6306576
```

Covariance is computed with the function `cov()`.

```
> cov(height, weight)
[1] 68.32596
```

Tip

This coefficient can also be computed with the function `cor.test()`.



```
> cor.test(height, weight)$estimate
      cor
0.6306576
```

11.5.4 Measures of Association Between a Quantitative Variable and a Qualitative Variable

11.5.4.1 Correlation Ratio $\eta_{Y|X}^2$

The correlation ratio $\eta_{Y|X}^2$ indicates how much of the variation of the quantitative variable Y is explained by the levels of the qualitative variable X with p levels. Indeed, X can be viewed as defining groups in the population. The correlation ratio is the ratio of variance within groups and inter-group variance. It is given by the formula

$$\eta_{Y|X}^2 = \frac{\sum_{k=1}^p n_k (\bar{y}_k - \bar{y})^2}{\sum_{i=1}^N (y_i - \bar{y})^2}$$

where n_k denotes the number of observations y_i corresponding to the k th level of X .

Here is the R code to compute this ratio:

```
> eta2 <- function(x, gpe) {
+   means <- tapply(x, gpe, mean)
+   counts <- tapply(x, gpe, length)
+   varwithin <- (sum(counts * (means - mean(x))^2))
+   vartot <- (var(x) * (length(x) - 1))
+   res <- varwithin/vartot
+   return(res)
+ }
```

We compute it for the variables `weight` and `gender`.

```
> eta2(weight, gender)
[1] 0.3325501
```

Tip

You can also use the following R instruction to compute $\eta_{Y|X}^2$:



```
> summary(lm(weight~gender))$r.squared
[1] 0.3325501
```

SECTION 11.6

Graphical Representations

The way a variable is represented graphically should always be adapted to the type of variable. Indeed, **the type of a variable often translates into specificities of a plot**. For instance, an arrow head at the end of an axis indicates that the corresponding levels are ordered. The direction of the axis of levels should thus be made evident for all variables, except those of a qualitative type. We have thus defined (and included in the package associated with this book) the function `arrowaxis()` which we shall use, when needed, to add an arrow on a plot axis. For some plots, we have decided to show side by side a basic version and a more elaborate and aesthetically pleasing version. We do not necessarily recommend using the elaborate version in data exploration, but these plots show the possibilities offered by R to modify a plot as one wishes.

11.6.1 Plotting Qualitative Variables

11.6.1.1 Cross Chart

For each observation, the cross chart shows a small horizontal bar in the column of the relevant level. This diagram is not included in R, but it can be coded with the function `plot()` and its argument `pch`. The corresponding function, which we call `crosschart()`, is included in the package associated with this book (Fig. 11.2).

Note that there also exists a function `dotchart()`. When coupled with the function `table()`, it gives a similar display, except that it is rather a representation of the table of counts of the variable (Fig. 11.3).

```
> crosschart(situation,col=c("orange","darkgreen","black","tan"))
```

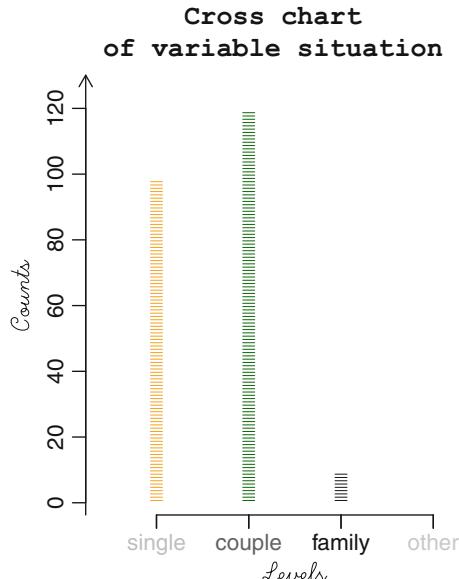


Fig. 11.2: Cross chart for a qualitative variable

```
> dotchart(table(situation),col=c("orangered","darkgreen",
+ "turquoise","tan"),pch=15,main=paste("Dot chart of counts",
+ "of variable situation",sep="\n"))
```

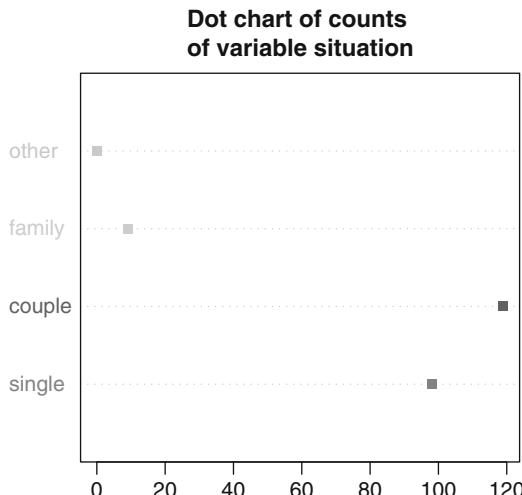


Fig. 11.3: Dot chart for a qualitative variable

11.6.1.2 Bar Charts

It is obtained with the function `barplot()`. To make the plot prettier, we propose the function `barchart()`, included in the package associated with this book (Fig. 11.4).

```
> col <- c("gray", "orangered", "lightgoldenrodyellow", "red")
> barplot(table(situation), col=col)
```

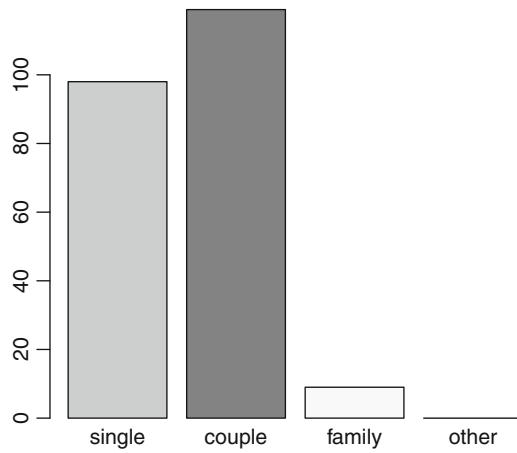
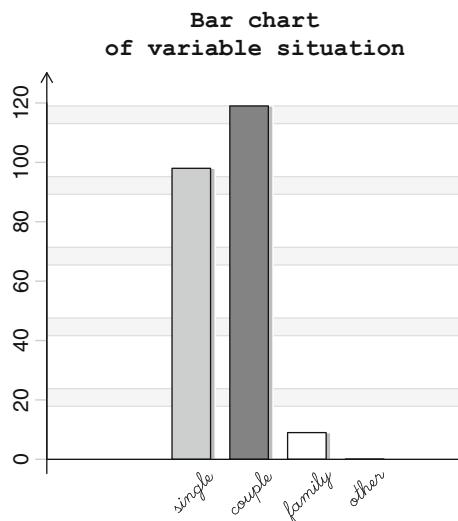


Fig. 11.4: Bar chart for a qualitative variable

```
> barchart(situation, col)
```



11.6.1.3 Pareto Chart

A Pareto chart can also be produced with the function `barplot()`, since it is a bar chart with bars ordered from tallest to smallest (Fig. 11.5).

```
> col <- c("yellow", "yellow2", "sandybrown", "orange",
+       "darkolivegreen", "green", "olivedrab2", "green4")
> barplot(sort(table(fat), TRUE), col=col)
```

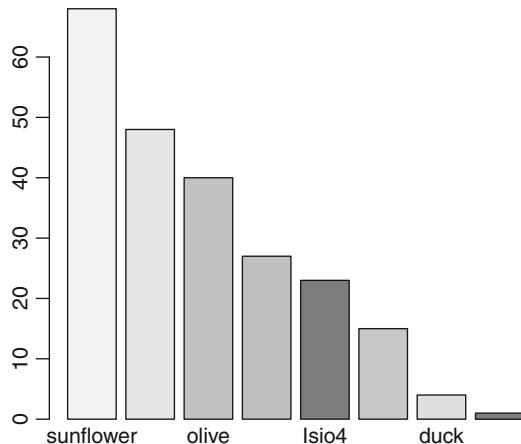
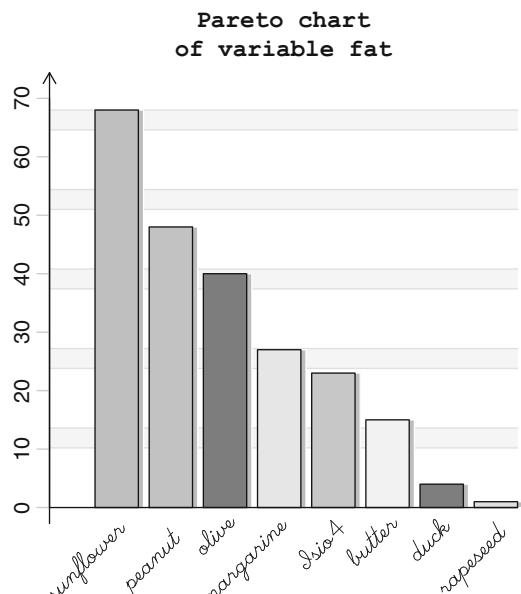


Fig. 11.5: Pareto chart for a qualitative variable

```
> barchart(fat, col, pareto=TRUE)
```



11.6.1.4 Stacked Bar Chart

A stacked bar chart can be obtained with the function `barplot()` with an object of type `matrix()` as first effective argument (Fig. 11.6).

```
> nbh <- table(gender)[1]
> nbf <- table(gender)[2]
> tabfreq.fat.males <- table(fat[gender=="Male"])/nbh
> tabfreq.fat.females <- table(fat[gender=="Female"])/nbf
> barplot(cbind(tabfreq.fat.males,tabfreq.fat.females),
+           main="Stacked bar chart for variable fat",col=
+           c("yellow","yellow2","sandybrown","orange",
+             "darkolivegreen","green","olivedrab2","green4"),xlim=
+             c(0,1),width=0.15,space=1,names.arg=
+             c("Males","Females"),legend=TRUE,density=40)
> arrowaxis(x=FALSE,y=TRUE)
```

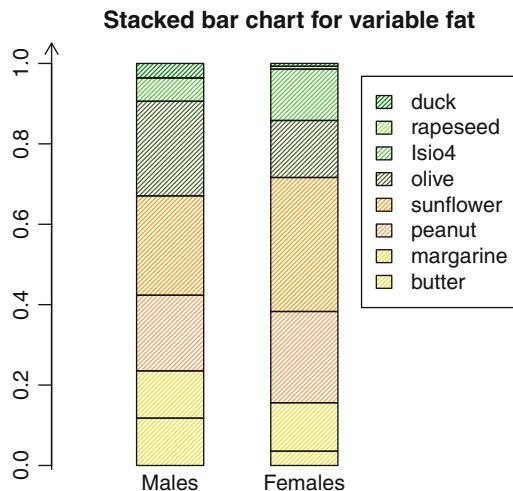


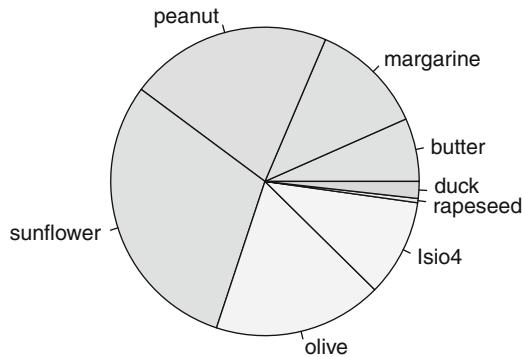
Fig. 11.6: Stacked bar chart for a qualitative variable

11.6.1.5 Pie Chart

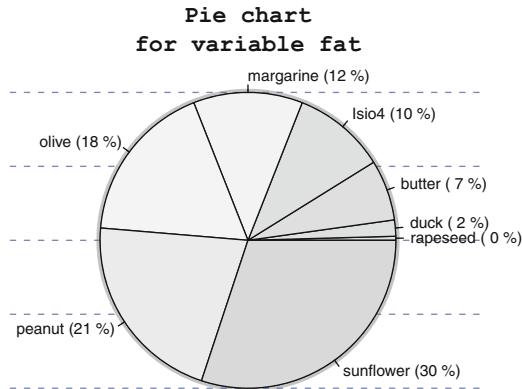
A pie chart can be obtained with the function `pie()`. We propose the more aesthetically pleasing function `camembert()`, included in the package associated with this book.

```
> require("RColorBrewer")
> col <- brewer.pal(8,"Pastel2")
```

```
> pie(table(fat), col=col)
```



```
> camembert(fat, col)
```



11.6.2 Plotting Ordinal Variables

11.6.2.1 Bar Chart With Cumulative Frequencies Line

Such a chart can be obtained with the functions `barplot()` and `points()` (Fig. 11.7). As previously, the function `barchart()` included in the package associated with this book allows a different representation (for instance, the line of cumulative frequencies is raised and projects a shadow).

```
> require("RColorBrewer")
> col <- brewer.pal(6,"Blues")
> tx <- table(fish)
> tx <- tx/sum(tx)
> r <- barplot(tx,ylim=c(0,1),col=col)
> points(r,cumsum(tx),type="l")
```

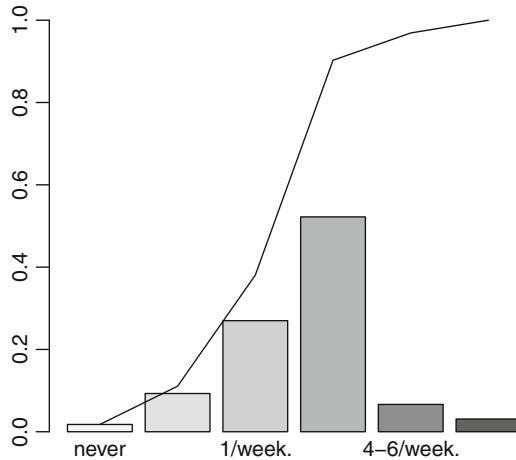


Fig. 11.7: Bar chart with cumulative frequencies line for an ordinal variable

11.6.3 Plotting Discrete Quantitative Variables

11.6.3.1 Cross Chart

A cross chart can be obtained with the function `crosschart()`, included in the package associated with this book.

See also

See Sect. 11.6.1, page 357.



11.6.3.2 Bar Chart

Such a chart can be obtained with the function `plot()` applied to a contingency table (Fig. 11.8).

11.6.3.3 Plotting the Empirical Distribution Function

Such a plot is obtained with the functions `plot()` and `ecdf()` (Fig. 11.9).

```
> plot(tabtea/length(unique(tea)), ylab="",  
+       col="darkolivegreen", lwd=5,  
+       main="Bar chart for variable tea")  
> arrowaxis()
```

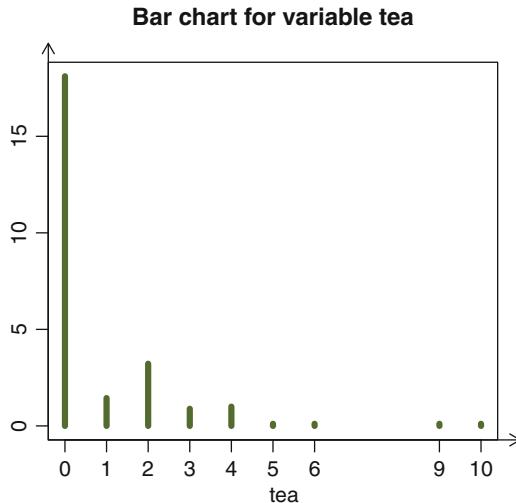


Fig. 11.8: Bar chart for a discrete quantitative variable

```
> plot(ecdf(na.omit(coffee)), main=paste("Empirical distribution  
function",  
+ "for variable coffee", sep="\n"), verticals=TRUE,  
+ ylab=expression(F[n](x)), col.01line="#89413A", col.points=  
+ "#6D1EFF", col.hor="#3971FF", col.vert='#3971FF')  
> arrowaxis(x=TRUE, y=TRUE)
```

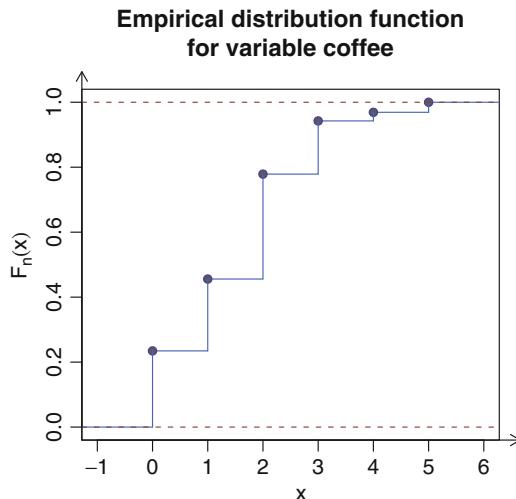


Fig. 11.9: Empirical distribution function for a discrete quantitative variable

11.6.3.4 Stemplot

A stemplot, or stem-and-leaf display, can be obtained with the function `stem.leaf()` of package `ap1pack`.

See also

See Sect. 11.6.4, page 367.



11.6.3.5 Boxplot

Such a chart can be obtained with function `boxplot()`, as shown below. The chart gives explanations (Fig. 11.10).

The box is drawn using the values of the three quartiles. Values represented as small circles are outliers, which might be suspect or deviant (`boxplot(cafe)$out`). These extreme values are those that are outside the box, further than 1.5 times the interquartile range (the argument `range` can be used to change this value from the default 1.5). Note that values which are outside the box, but within 1.5 times the interquartile range, are called adjacent values. The whiskers are drawn at the largest and smallest adjacent value.

Tip

Type `example(bxp)` or `example(boxplot)` for other examples of such diagrams.



```
> boxplot(coffee,col="orange",
+ main="Boxplot for variable coffee")
> arrowaxis(x=TRUE,y=TRUE)
```

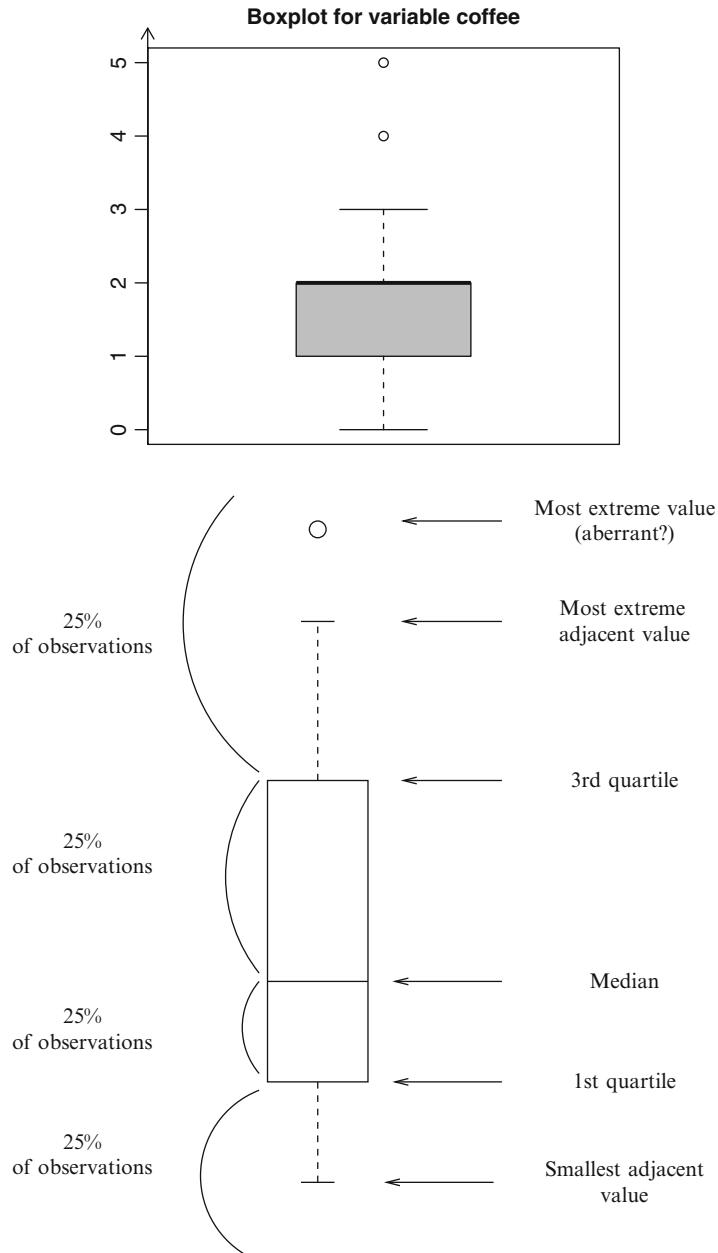


Fig. 11.10: Boxplot and explanations

11.6.4 Plotting Continuous Quantitative Variables

We now present a few useful graphs for exploring quantitative data.

11.6.4.1 Empirical Distribution Function

Use the functions `plot()` and `ecdf()` (Fig. 11.11).

```
> plot(ecdf(na.omit(age)), main=paste("empirical distribution
  function",
+ "of variable age", sep="\n"), col.hor='# 3971FF', col.points
= '#3971FF')
> arrowaxis()
```

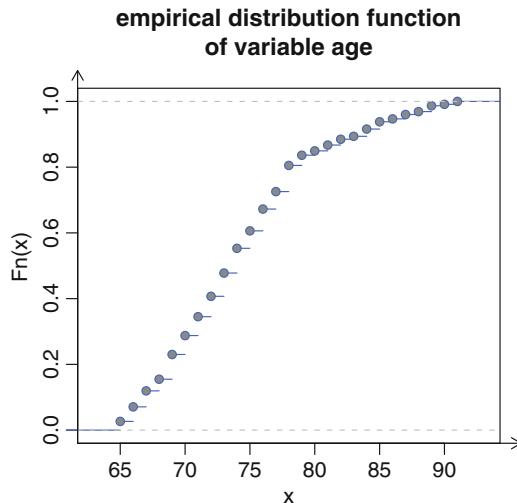


Fig. 11.11: Plot of the empirical distribution function for a continuous quantitative variable

11.6.4.2 Stemplot

You can use the function `stem()`, but it is not very sophisticated. We recommend the function `stem.leaf()` from package `aplypack`. Chart construction is done in three steps:

- Choose a leaf unit, using argument `unit`.
- Choose the number of parts (1, 2 or 5) in which each stem should be divided, using argument `m`.
- Choose the representation style, using argument `style` which can take the values "Tukey" or "bare".

```
> require("aplypack")
> stem.leaf(height,m=5,style="bare")
1 / 2: represents 12
leaf unit: 1
      n: 226
 1    14 | 0
     14 |
     14 |
     14 |
     14 |
 3    14 | 88
12    15 | 000001111
24    15 | 222223333333
45    15 | 4444444444445555555555
60    15 | 6666666666667777
73    15 | 8888888899999
97    16 | 00000000000000000000011
(22)   16 | 2222222223333333333333
107   16 | 4444444555555555555555
 85   16 | 6666777
 78   16 | 88888888888999
 64   17 | 000000000011111
 49   17 | 2222222222333
 36   17 | 444555555
 27   17 | 6666666777
 17   17 | 88889
 12   18 | 0011
   8   18 | 22
     18 |
   6   18 | 666
   3   18 | 888
```

11.6.4.3 Boxplots

They can be created with the function `boxplot()`.

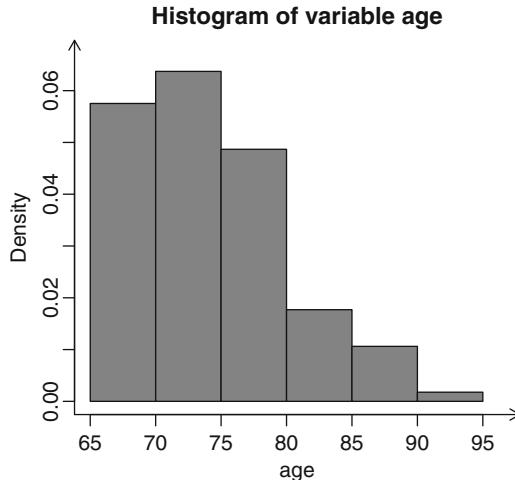
See also

See Sect. 11.6.3, page 365.

11.6.4.4 Density Histogram with Identical or Different Class Ranges

Use the function `hist()` (Fig. 11.12).

```
> classes <- hist(age,right=TRUE,freq=FALSE,ylab="Density",
+ main="Histogram of variable age",col="orangered")
> arrowaxis()
```



```
> classes <- hist(weight,right=TRUE,freq=FALSE,
+ main="Histogram of variable weight",
+ ylab="Density",breaks=c(min(weight),50,80,
+ 90,max(weight)),col="olivedrab")
> arrowaxis()
```

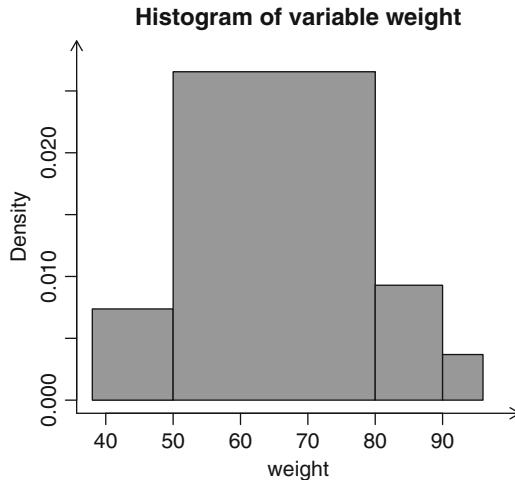


Fig. 11.12: Density histogram with identical or different class ranges

11.6.4.5 Frequency Polygon

Use the functions `hist()` and `segments()` (Fig. 11.13).

```

> classes <- hist(height,right=TRUE,freq=FALSE,
+ main=paste("Histogram and frequency polygon",
+           "of variable height",sep="\n"),col="orangered")
> middles <- classes$mid ; mlon <- length(middles)
> densities <- classes$density
> segments(middles[1:mlon-1],densities[1: mlon-1],
+            middles[2:mlon],densities[2:mlon],col=
+            rgb(0.4196078,0.4196078,0.1372549,0.9),lwd=3)
> arrowaxis()

```

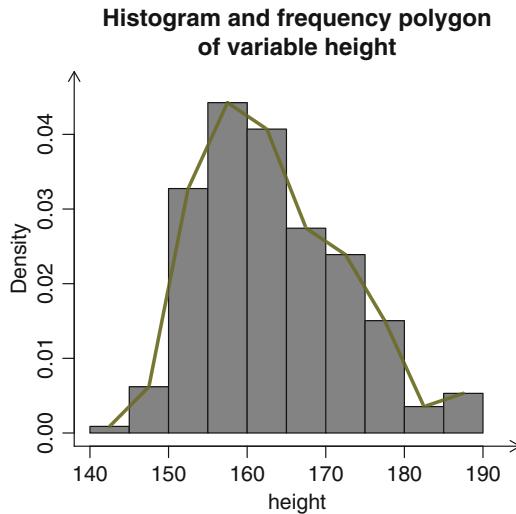


Fig. 11.13: Frequency polygon

11.6.4.6 Cumulative Frequency Polygon

Use the functions `hist()`, `ecdf()` and `plot()` (Fig. 11.14).

```

> bounds <- hist(height,right=TRUE,plot=FALSE)$breaks
> plot(bounds,ecdf(height)(bounds),type="l",main=
+       paste("Cumulative frequency polygon","of variable height",
+             sep="\n"),
+       ylab="Frequencies",col="darkolivegreen",lwd=3)
> arrowaxis()

```

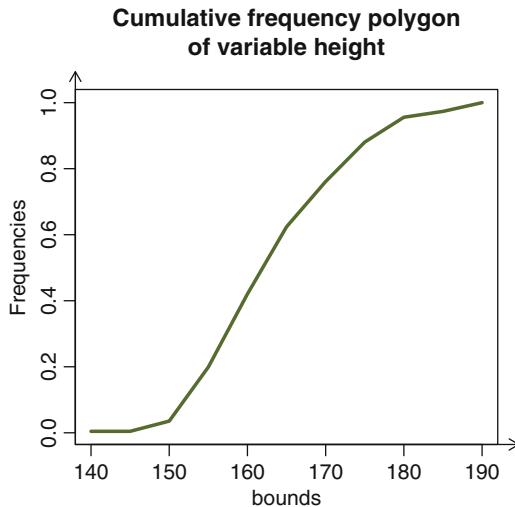


Fig. 11.14: Cumulative frequency polygon

Tip

This plot can be used to give a graphical estimator of the median. Simply add a horizontal line at height $h = 0.5$ with the instruction `abline(h=0.5)`. Then, type the instruction `locator(1)$x` and click on the intersection between this horizontal line and the cumulative frequency polygon. Similarly, you can get any other quantile by replacing the value 0.5 with the desired quantile.



11.6.5 Graphical Representations in a Bivariate Setting

In this section, we present a few useful representations for bivariate settings.

11.6.5.1 Two-Way Plots for Two Qualitative Variables

You can overlay two barplots, as shown on the next two figures (Fig. 11.15).

```
> tss <- prop.table(table(gender,situation),1)
> barplot(tss,bes=TRUE,leg=TRUE)
> title(paste("Barplots of situation","as a function of gender",
+ sep="\n"))
> arrowaxis(FALSE,TRUE)
```

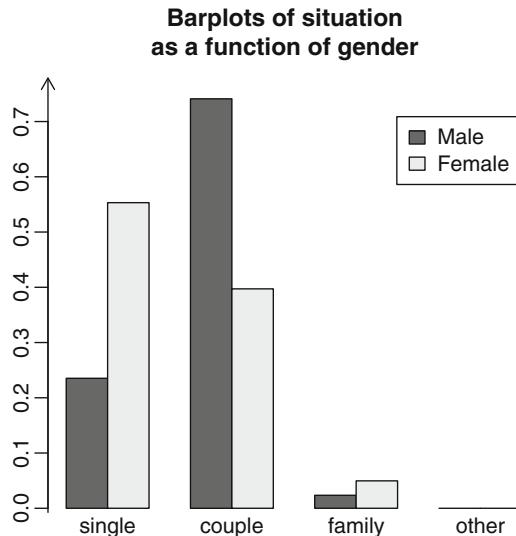


Fig. 11.15: Bar plot for two qualitative variables

A mosaic plot can also be useful to visualize the relationship between two qualitative variables (Fig. 11.16).

```
> par(las=1) # Horizontal writing of levels.
> mosaicplot(gender~fat,color=brewer.pal(5,"Set1"),
+             main="Mosaicplot of fat as a function of age")
```



Fig. 11.16: Mosaic plot for two qualitative variables

Tip

Another interesting function in this setting is `assocplot()` which produces a Cohen–Friendly association plot indicating deviations from independence in a 2×2 contingency table (Fig. 11.17). We can also cite the function `spineplot()`.



```
> assocplot(table(gender,fat))
```

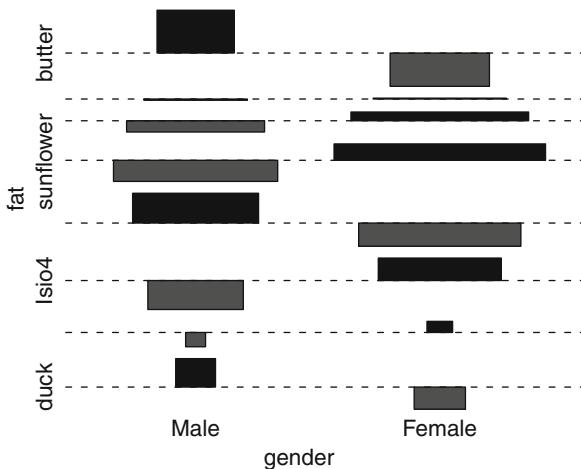


Fig. 11.17: Cohen–Friendly association plot for two qualitative variables

One last interesting function is `table.cont()` from package `ade4` (Fig. 11.18).

```
> require("ade4")
> genderfat <- table(gender,fat)
> table.cont(genderfat, row.labels=rownames(genderfat),
+             col.labels=colnames(genderfat))
```

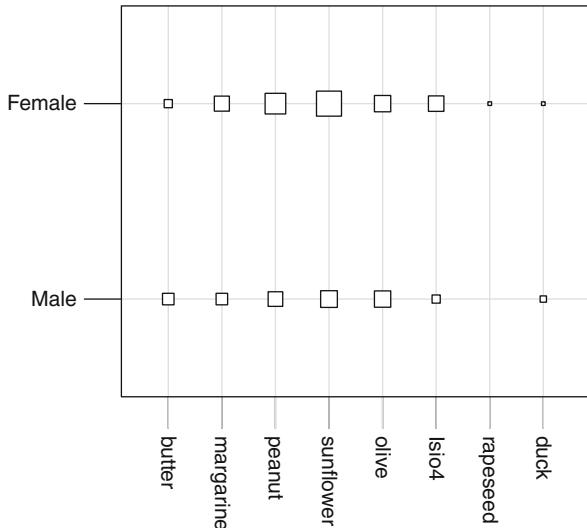
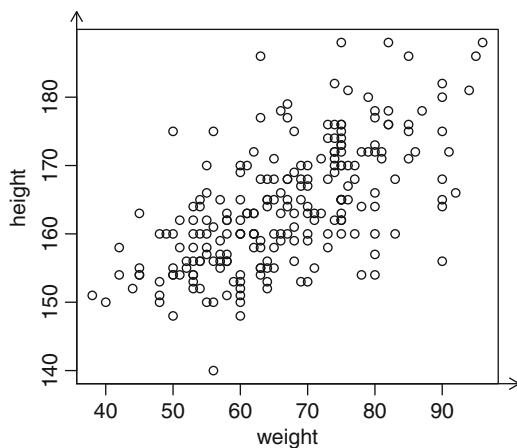


Fig. 11.18: `table.cont` plot for two qualitative variables

11.6.5.2 Two-Way Plots for Two Quantitative Variables

In this setting, use the function `plot()`.

```
> plot(height~weight)
> arrowaxis()
```



We have seen in Chap. 7 how to make this plot prettier. To this end, we create a function `flashy.plot()` (Fig. 11.19).

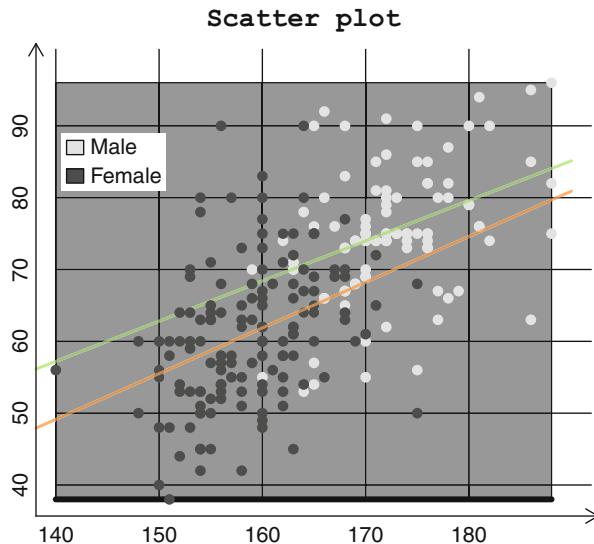


Fig. 11.19: Plot of two quantitative variables

11.6.5.3 Two-Way Plots for One Qualitative and One Quantitative Variable

In this setting, it is interesting to draw box plots of the quantitative variable for each level of the qualitative variable. If the variables are structured correctly in R, you simply need to call the function `plot()` (Fig. 11.20).

```
> par(bty="n")
> plot(coffee~gender,col=brewer.pal(5,"Set2"),
+ notch=TRUE,varwidth=TRUE,boxwex=0.3)
> title(paste("Boxplot of coffee consumption",
+ "as a function of gender",sep="\n"),family="Courier")
> arrowaxis(F,T)
```

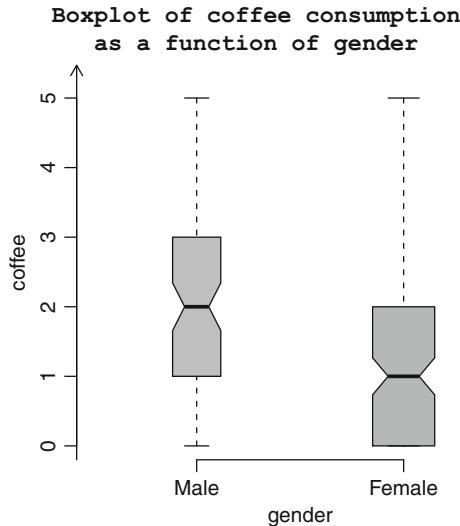


Fig. 11.20: Box plots of a quantitative variable as a function of the levels of a qualitative variable

Also note that the function `stripchart()` is useful to get the following plot (Fig. 11.21):

```
> stripchart(raw_fruit~age,data=nutrielderly)
> arrowaxis(y=TRUE)
```

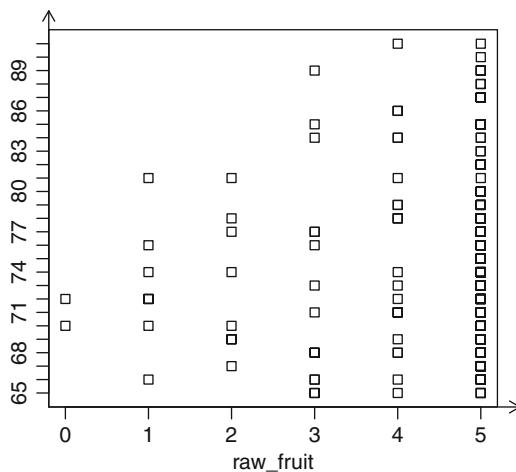


Fig. 11.21: `stripchart` plot for a quantitative and a qualitative variable

Memorandum

`as.factor()`: transform a variable into factors
`levels()`: display or affect the levels of a factor
`as.ordered()`: transform a variable into ordered factors
`as.integer()`: structure a discrete variable
`as.double()`: structure a continuous variable
`table()`: table of counts for a variable or contingency table between two variables
`addmargins()`: add margins to a contingency table
`margin.table()`: marginal distributions of a contingency table
`prop.table()`: conditional distributions from a contingency table
`na.omit()`: remove missing values (NA) of a variable
`median()`: median of a vector
`mean()`: mean of a vector
`quantile()`: quantiles of a vector
`summary()`: when applied to a numerical series, returns minimum, maximum, quartiles and mean
`range()`: minimum and maximum
`IQR()`: interquartile range
`var()`: variance of a sample
`sd()`: standard deviation of a sample
`mad()`: absolute deviation from the median
`chisq.test()`: chi-squared statistic
`cor.test()`: Pearson's correlation coefficient, Kendall's τ or Spearman's ρ
`cov()`: covariance
`cor()`: correlation
`barplot()`: draw a bar chart
`pie()`: draw a pie chart
`plot.ecdf()`: plot the empirical cumulative distribution function
`stem()`: draw a stem plot
`boxplot()`: draw a box plot
`hist()`: draw a histogram
`plot()`: draw a scatter plot



Exercises

- 11.1-** Give the instruction which returns the table of frequencies for a qualitative variable x .
- 11.2-** Give the instruction which returns the contingency table for qualitative variables x and y .
- 11.3-** Which function returns the marginal distributions from a contingency table?
- 11.4-** Which function returns the conditional distributions from a contingency table?
- 11.5-** Give the instruction which returns the mode of a distribution.
- 11.6-** Give the instruction which returns the range of a vector x .
- 11.7-** Give the instruction which returns the interquartile range of a vector x .

- 11.8-** Give the instruction which returns the (nonempirical) variance of a vector \mathbf{x} .
- 11.9-** Give the code of a function which calculates the coefficient of variation.
- 11.10-** Give the instruction which calculates the mean absolute deviation.
- 11.11-** Which package includes functions to calculate skewness and kurtosis?
- 11.12-** Give the instruction which returns Cramér's ϕ^2 .
- 11.13-** Give the code of a function which calculates the correlation ratio $\eta_{Y|X}^2$.
- 11.14-** Which function would you use to draw a Pareto chart?
- 11.15-** Which function would you use to draw a stacked bar chart?
- 11.16-** Which function would you use to draw a pie chart?
- 11.17-** Which function would you use to draw a box plot?
- 11.18-** Which function would you use to draw a histogram?



Worksheet

Descriptive Data Studies

A - Thoughts on Independence in Descriptive Statistics

- 11.1-** Import the file http://www.biostatisticien.eu/springer/snee74_en.txt into an R object called `snee`.
- 11.2-** Display the first and last lines of `snee` with the functions `head()` and `tail()`. How many individuals are there? How many variables? What type are the variables?
- 11.3-** Use the function `attach()` on your `data.frame`, then check with the functions `class()` and `levels()` that the structure of your variables is correct. What are the levels of the variables?
- 11.4-** Perform a univariate descriptive study of each variable: numerical results and appropriate graphical representations.
- 11.5-** We shall now study the dependence of variables `eyes` and `hair`. Create the contingency table `eyeshair` (observed counts) of variables `eyes` and `hair`.
- 11.6-** Calculate the frequency of each level of the variable `hair` (profile by column). You get the distribution function `fhair` of hair colour in the population.
- 11.7-** Now, include the second characteristic: eye colour. Calculate the number `nblue` of individuals with blue eyes in the entire population.
- 11.8-** Suppose that eye colour and hair colour are independent. In other words, the fact that an individual has blue eyes bears no relation with the colour of their hair. In that case, the proportions calculated in 11.6 should still be the same within the subpopulation of people with blue eyes. Under this independence hypothesis, calculate the number of blue-eyed people who should have blond hair (respectively, brown, black and red).
- 11.9-** Do the same with other eye colours. You get a table `tab.ind1` of theoretical counts under the hypothesis of independence between eye and hair colour.

- 11.10-** Repeat the entire process, but with the two characteristics inverted (i.e. start with the variable `eyes`). From table `eyeshair`, calculate the frequency of each level of the variable `eyes` (profiles by rows). You get the distribution function `feyes` of eye colours in the population.
- 11.11-** Now include the second characteristic: hair colour. Calculate the number `nblond` of people with blond hair in the entire population.
- 11.12-** Suppose that hair and eye colour are independent. In other words, the fact that an individual has blond hair bears no relation with their eye colour. In that case, the proportions calculated in 11.10 should still be the same within the subpopulation of people with blond hair. Under this independence hypothesis, calculate the number of blond people who should have blue eyes (respectively, brown, hazel and green).
- 11.13-** Do the same with other hair colours. You get a table `tab.ind2` of theoretical counts under the hypothesis of independence between hair and eye colour.
- 11.14-** Use the function `a11.equal()` to compare the two tables of theoretical counts. What do you observe?
- 11.15-** Compare the table of observed counts `eyeshair` with the table of theoretical counts (for each entry, calculate the square of the difference).
- 11.16-** Calculate the table of contributions to the χ^2 .
- 11.17-** Calculate all relevant link indicators. Conclude.
- 11.18-** Independence can also be defined (and this is in fact the primary definition) as equality of all conditional distributions. Calculate the conditional distribution of the variable `hair` knowing that the eye colour is blue (i.e. the frequency of various hair colours knowing that eye colour is blue). Calculate the three other conditional distributions of the variable `hair`. Calculate the conditional distributions of the variable `eyes` knowing each of the levels of variable `hair`. Conclude.
- 11.19-** Perform a descriptive study of variable `hair` as a function of variable `gender`.
- 11.20-** Perform a descriptive study of variable `eyes` as a function of variable `gender`.
- 11.21-** Analyze the dependence between eye colour and hair colour for the following contingency table:

		Hair				Total
		Blond	Brown	Black	Red	
Eyes	Blue	1,768	807	189	47	2,811
	Grey/green	946	1,387	746	53	3,132
	Brown	115	438	288	16	857
Total		2,829	2,632	1,223	116	6,800

B- Descriptive Analysis of Data Set NUTRIELDERLY

We now propose to solve the following questions on the data set NUTRIELDERLY, in which deliberate errors were introduced:

- 11.1-** Import the data file nutrition_elderly.xls.
- 11.2-** Give the absolute mode of variables situation, chocol and height.
- 11.3-** Choose classes for variable height and give the modal class.
- 11.4-** Calculate the median of variable chocol.
- 11.5-** Give frequency tables of variables chocol and raw_fruit.
- 11.6-** Using **only** these frequency tables, give the median of these two variables.
- 11.7-** Calculate the quartiles of variable height using the classes defined earlier.
- 11.8-** Draw the cumulative frequency polygon for variable height. On this plot, estimate the quartiles of the distribution.
- 11.9-** Using individual data, calculate the mean of variables height, weight and age.
- 11.10-** Calculate the frequency table of variable tea. Using this table, calculate the mean of this variable.
- 11.11-** Calculate the mean of variable height using the classes defined earlier.
- 11.12-** Calculate the range of variable weight.
- 11.13-** Draw a boxplot of variable weight.
- 11.14-** Using individual data, calculate the standard deviation of variable height.
- 11.15-** Using **only** its frequency table, calculate the coefficient of variation of variable tea.
- 11.16-** Calculate the total variance, the within-group variance and the between-group variance of variable coffee with the population split into two groups: males and females. Calculate the coefficient η^2 .

C- Descriptive Analysis of Data Sets

- 11.1-** Perform a descriptive statistical analysis of data set BIRTH-WEIGHT.
- 11.2-** Perform a descriptive statistical analysis of data set INFARCTION.

Chapter 12

A Better Understanding of Random Variables, Distributions and Simulations Using R Specificities

Goals of this chapter

We use the specificities of R to build empirically the notions of random variable, distribution of a random variable, law of large numbers and central limit theorem. We introduce some complex notions for statistical inference and examine sampling variation as well as the bias and variance of estimators. We go on to describe a few classical methods for simulating from a distribution. At the end of the chapter, we give commands to generate observations from common probability distributions and to calculate their probability and cumulative distribution functions and quantiles.

— SECTION 12.1 —

Notions on Random Number Generation

Consider an urn containing n balls numbered 1 to n . Random number construction can be imagined as an experience in which we pick (or *draw*) with replacement a single ball at a time, several times in a row. This operation produces a sequence of integers. The order in which these numbers appear is governed by a law called the discrete uniform distribution (on the set $\{1, \dots, n\}$). It is then natural to wonder how to generate real numbers uniformly over an interval. We introduce a few elements on number generation from such a uniform distribution, using a computer algorithm.

Random number generation is an essential part of simulation. The production of such numbers can be based on a mathematical algorithm which imitates randomness. We now explore such an algorithm ([34]), based on linear congruence, with period $2^{31} - 1$.

The algorithm is defined by the prime number $m = 2^{31} - 1$, by an initial value x_1 chosen arbitrarily in $\{1, \dots, m - 1\}$ and called the seed of the generator and by the following recurrence relation which produces n values (seed included):

$$x_{j+1} \leftarrow 48271 \times x_j \pmod{m}, \quad j = 1, \dots, n - 1$$

followed by a normalization (to the unit interval) step:

$$x_{j+1} \leftarrow x_{j+1}/m, \quad j = 1, \dots, n - 1.$$

Recall that $x \pmod{m}$ represents the “remainder in the division of x by m ”.

Note

Other similar algorithms are presented in Robert and Casella’s book ([35]).

Do it yourself



Give the sequence of R instructions which code this algorithm to create $n = 30$ values. Recall that $x \pmod{m}$ can be computed in R with the instruction `x%%m`.

The following vector `x` contains $n = 30$ values generated from this algorithm.

```
> x
[1] 0.2785000 0.4735070 0.6554026
[4] 0.9377200 0.6813453 0.2190502
[7] 0.7743987 0.9984000 0.7660816
[10] 0.5251324 0.6643606 0.3513046
[13] 0.8235011 0.2202823 0.2478704
[16] 0.9542908 0.5703867 0.1386948
[19] 0.9379738 0.9309887 0.7576328
[22] 0.6942503 0.1547888 0.8092940
[25] 0.4322561 0.4344264 0.1965560
[28] 0.9561434 0.9964051 0.4690752
```

As you can see, the values created by this algorithm are unpredictable (apart from the fact that they are between 0 and 1). However, because they were created with a deterministic mathematical algorithm, these values are called **pseudo-random** numbers.

The algorithm above is a pseudo-random number generator called **uniform** over $[0, 1]$, because it generates numbers spread uniformly over this interval.

Tip

Pseudo-random number generation is of course implemented at the core of R. For instance, a more elaborate version of the algorithm above is available through the R function `runif()`. The function `set.seed()` is used to set the seed of the algorithm.



SECTION 12.2

The Notion of Random Variables

In the previous section, we proposed an algorithm to generate pseudo-random numbers. In probability and statistics, such a process is usually called a random variable. The purpose of this section is to propose a heuristic on this topic: a random variable could be viewed as an algorithm to generate numbers, which are then called realizations of the random variable.

12.2.1 Realizations of a Random Variable and Functioning Law

In this section, we shall see how the syntax of R can help us better understand the difference between a random variable and its realizations.

A common example is that a statistics problem starts with one of the following assertions: (1) “Let X be a random variable with distribution $\mathcal{U}(0, 1)$ ” or (2) “Let X be a random variable with distribution $\mathcal{N}(0, 1)$ ”.

A **random variable** can be viewed as a **number creation factory** or, in mathematical terms, as a function which creates a random number every time it is called. The process which creates values generated by a random variable is governed by a **functioning rule** imposed upon this variable, called the **law of the random variable**. This is shown clearly in the body of the function that defines it.

In R, it is easy (and informative) to create such random variables. For instance, the instruction

```
> X <- function() runif(1)
```

corresponds to assertion (1). Successive calls of this function create **realizations** of the random variable X:

```
> X()
[1] 0.8806583
> X()
[1] 0.07355798
> X()
[1] 0.05503209
```

Similarly, the instruction

```
> X <- function() rnorm(1)
```

corresponds to assertion (2). Once again, successive calls of this function create realizations of the new random variable X:

```
> X()
[1] 0.6683035
> X()
[1] -0.3708295
> X()
[1] 0.7179792
```

We find this makes it easier to understand the difference between a **random variable** (the function) and its **realizations** (the numbers output by the function). As you can see, a random variable is a “machine” (a procedure) which makes values, called realizations. It is random in the sense that the realizations change at each call, and that it is impossible to predict which value will be output.

Warning

Note that the two random variables we have introduced are built in the same way (the instructions are the same), except for the **functioning rule or law** which governs them (`rnorm` or `rnorm` in the body of the function). A synonymous often used for **law** is **distribution**. This comes from the fact that the cumulative distribution function (seen later), which describes how the values produced by a random variable are distributed, entirely characterizes the law of this variable.



12.2.2 I.i.d. Random Variables

Another standard statement in statistics is: “Let X_1, \dots, X_n be n independent and identically distributed (i.i.d.) random variables following the distribution $\mathcal{N}(0, 1)$ ”. This concept can easily be translated into R. For example, with $n = 4$, we get

```
> X <- function() rnorm(1) # r.v. governed by the distribution
   # rnorm (i.e. N(0,1)).
> X4 <- X3 <- X2 <- X1 <- X      # Translates the fact that
   # X_i are i.i.d.
```

Warning

X_1 and X_2 follow the same distribution `rnorm`. They are identically distributed. However, they are not identical, in the sense that they do not produce the same values. Independence of X_1 and X_2 means the production of values by X_1 is not influenced by X_2 :

```
> X1
function() rnorm(1) # r.v. governed by the Normal distribution
> X2
function() rnorm(1) # r.v. governed by the Normal distribution
> c(X1(),X2())
[1] -0.6319022 -0.1029233
```



Note that you could also consider the vector $\mathbf{X} = (X_1, \dots, X_4)$, which is a random vector with four components and which produces four “independent” realizations simultaneously at each call.

```
> vecX <- function() c(X1(),X2(),X3(),X4())
> vecX()
[1] 0.8041706 -3.6413500 1.7871627 0.7710802
> # or, equivalently in R:
> vecX <- function() rnorm(n = 4)
> vecX()
[1] -0.3010423 0.7895389 0.2059419 1.0169773
```

Tip

If the n X_i are i.i.d. with same distribution as X , then $\mathbf{X}_n = (X_1, \dots, X_n)$ can be created using

```
> vecXn <- function(n) replicate(n,X())
```



The random vector $\mathbf{X}_n = (X_1, \dots, X_n)$ is called the **sample**.

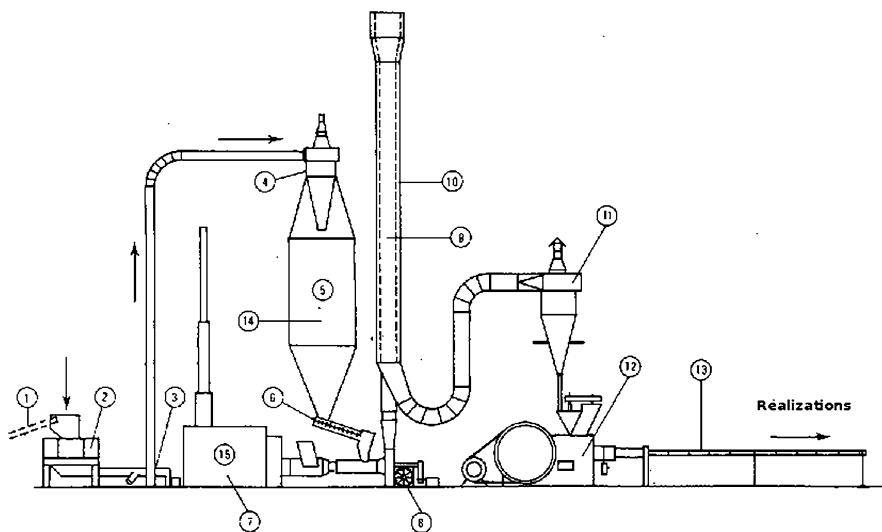
12.2.3 Characterizing the Distribution of a Random Variable

We have seen that the process by which a random variable produces values is governed by a **functioning rule**, called the **distribution** of the random variable.

Note

We shall use the standard symbol \sim to indicate that a random variable X follows a given distribution. For example, we shall write $X \sim \mathcal{U}(0, 1)$ to indicate that the random variable X follows a uniform distribution over the interval $[0, 1]$ or $X \sim \mathcal{N}(0, 1)$ to indicate that X follows a standard normal distribution.

The next drawing illustrates the idea that a random variable is a “machine” governed by **parameters** and which outputs realizations. There is a blueprint of the machine, which governs how the realizations are created. But slight, unpredicted variations in the run of the process lead to variations in the output, which can therefore not be completely predicted. Indeed, each specific realization is unpredictable and governed by randomness. However, since there is a fixed functioning rule for the generator, it is still possible to describe a few global properties of these numbers. There are thus several kinds of randomness, each structured by the distribution of the random variable at play. When speaking of “random numbers”, it is necessary to specify the kind of randomness.



Mathematicians have introduced mathematical objects describing this structure: the probability density function (or density or p.d.f.); the cumulative distribution function (or distribution function or c.d.f.); and the quantile function of a random variable, to name a few. In most cases, these functions characterize completely the distribution of the random variable.

12.2.3.1 Density Function, Distribution Function and Quantile Function

Suppose that we observe several unpredictable values produced by a random variable X . We wish to **describe** these values and, through them, to better understand or specify the functioning of the generator X .

Here are a few examples of what you might notice for a given random variable X :

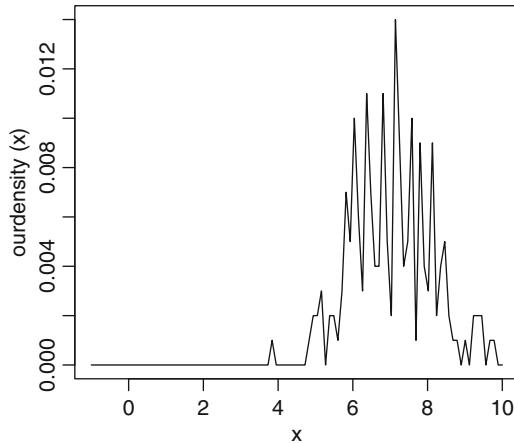
- there are positive and negative values;
- they are gathered around the number 7;
- as many values are greater than and lesser than 7;
- they are spread symmetrically about 7.

On a more global scale, you can also look at the **density** of observations (in the common meaning of the word: a high density corresponds to values close to each other) generated by X in intervals $[x - \epsilon, x + \epsilon]$ (where ϵ is a very small non-negative number) of the range of possible values. The range of possible values is called the **support** of X . The code of the function **density**, given below, was written for didactic reasons and is not efficient; it should help translate the notion of density of observations.

```
> # The code (distribution) of the r.v. X is temporarily hidden
# on purpose.
> X <- function() some.code(some,parameters)
> ourdensity <- function(x,n=1000,eps=0.01) {
+   lx <- length(x)
+   res <- vector("integer",lx)
+   obs <- replicate(n,X()) # Generate observations from X.
+   for (i in 1:lx) { res[i] <-
+     length(which((x[i]-eps) <= obs & obs <= (x[i]+eps)))/n
+   }
+   return(res)
+ }
```

The plot below represents the function which associates to each interval in the support of X the value y of the relative concentration of observations in this interval (i.e. the frequency of values belonging to this interval).

```
> curve(ourdensity,xlim=c(-1,10))
```



If you increase the number n of observations generated by X and decrease ϵ , this plot becomes smoother.

```
> curve(ourdensity(x,n=1000000,eps=0.001),xlim=c(-1,10))
```

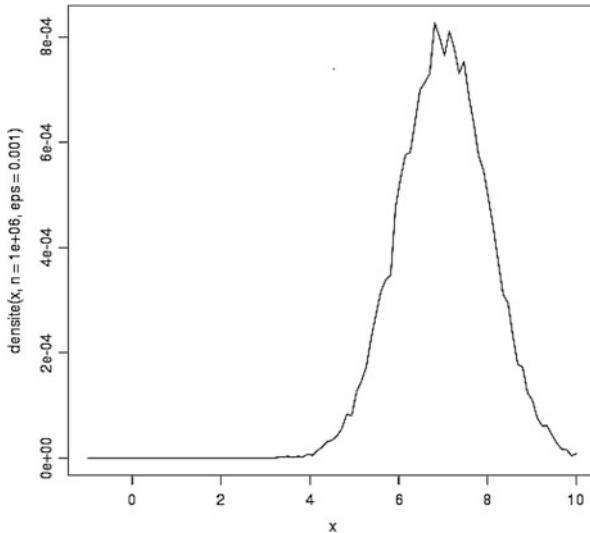


Fig. 12.1: Plot approximating the density of X

In the limit, as $n \rightarrow \infty$ and $\epsilon \rightarrow 0$, this plot becomes perfectly smooth and regular. Once normalized so that its area equals 1, it is called the **density function** of the random variable X and is denoted f_X .

Note

A random variable whose possible values can make a continuous set of values (such as an interval) is called **continuous**. A random variable which can only take a finite or countable number of different values (also called modalities) is called **discrete**. For a discrete random variable, we do not speak of its density, but of its **mass function** which gives the probability of outcome of each modality.



To sum up, it could be said that describing the **distribution** of a random variable X consists in giving two pieces of information:

- (1) The range of possible values or **support** of X (which can be discrete or continuous)
- (2) For each (infinitesimal) interval in this range, the value of the **density** of X for a continuous random variable or of the **mass function** of X for a discrete random variable

Note

The **cumulative distribution function** $F_X(x)$, which accumulates the densities of observations up to the point x (or which accumulates the observation probabilities of all modalities up to and including x for a discrete variable), is another way of characterizing the distribution of a random variable. It can be shown that it represents the area of the density plot f_X up to x and that it gives the probability that the random variable X creates observations smaller than x :

$$F_X(x) = P[X \leq x] = \int_{-\infty}^x f_X(t)dt.$$

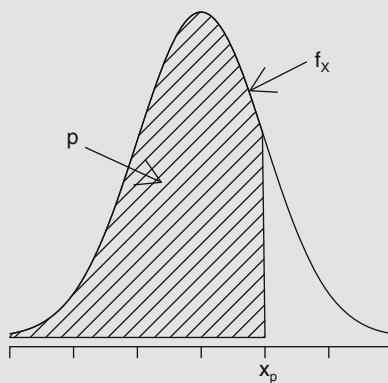


The **quantile function** F_X^{-1} , the inverse distribution of the cumulative distribution function, also characterizes a random variable.

For each probability value $p \in [0, 1]$, the value $x_p = F_X^{-1}(p)$ is called the **fractile or quantile of order** p of random variable X . It is defined by the equation

$$x_p = F_X^{-1}(p) \Leftrightarrow F_X(x_p) = P[X \leq x_p] = p.$$

Thus, the probability that the realizations of the random variable X be lesser than or equal to the value x_p is p . The following plot illustrates this notion.



12.2.4 Parameters of the Distribution of a Random Distribution

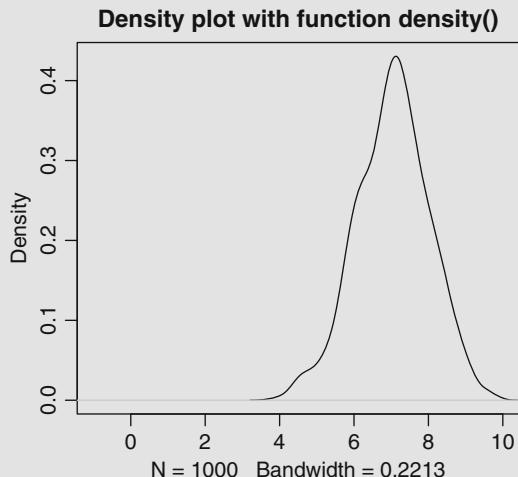
We had deliberately hidden the body of the function defining the random variable X . We can now reveal that the density shown on Fig. 12.1 is the density of a random variable X following the distribution $\mathcal{N}(\mu = 7, \sigma^2 = 1)$.

```
> X <- function() rnorm(1,7,1)
```

Tip

The R function `density()` gives the density plot of a sample of observed values.

```
> plot(density(rnorm(1000,7,1)),xlim=c(-1,10),
+ main="Density plot with function density()")
```



In this definition, the quantities $\mu = 7$ and $\sigma^2 = 1$ appear explicitly. They are called the **parameters** of this distribution.

Let X_1, \dots, X_n be n independent and identically distributed (i.i.d.) random variables following the normal distribution with mean $\mu = 7$ and variance $\sigma^2 = 1$. We also define $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$.

We have noticed that it is sometimes difficult to understand the (fundamental) difference that exists between:

- The random variable \bar{X}_n
- Its realizations \bar{x}_n
- The **parameter** $\mu = \mathbb{E}(X)$, which is the theoretical expectation

It is true that, by abuse of language, these three objects are often all called by the same name: *the mean*.

We believe that the R language helps better understand and distinguish these concepts. The random variable $\bar{X}_4 = \frac{1}{4} \sum_{i=1}^4 X_i$ (for $n = 4$) could be defined as

```
> X1 <- function() rnorm(1, mean = mu <- 7, sd = 1)
> X4 <- X3 <- X2 <- X1
> Xbar4 <- function() (X1() + X2() + X3() + X4()) / 4
```

Advanced users

The random variable \bar{X}_n can be defined automatically (for small n):

```
> n <- 10
> eval(parse(text=paste(paste("X", 1:n, " <- ", sep="", 
+                                     collapse=""), "X")))
> eval(parse(text=paste("Xbar", n, " <- function() 
+   (", paste("X", 1:(n-1), " () +", sep="", collapse=""),
+   "X", n, " ()) /", n, sep="")))
> Xbar10
function()
(X1() + X2() + X3() + X4() + X5() + X6() + X7() + X8() + X9() + X10()) / 10
```



This function can be used repeatedly to create several realizations, which could be denoted as $\bar{x}_4^{(i)}, i = 1, 2, \dots$:

```
> xbar4.1 <- Xbar4()
> xbar4.1
[1] 7.495727
> xbar4.2 <- Xbar4()
> xbar4.2
[1] 7.069361
> xbar4.3 <- Xbar4()
> xbar4.3
[1] 6.837127
```

It is clear that successive calls of the same function `Xbar4()` result in several different unpredictable realizations which **vary** around 7: the $\bar{x}_4^{(i)}$. This helps show that:

- \bar{X}_4 is indeed a random variable, i.e. a machine to create realizations;
- the realizations $\bar{x}_4^{(1)}, \bar{x}_4^{(2)}, \bar{x}_4^{(3)}, \dots$ are all different and unpredictable and come from the random variable \bar{X}_4 ;
- the machine \bar{X}_4 is made of the individual random variables X_1, \dots, X_4 , which all follow the same functioning law;
- this functioning law depends on a parameter `mu` (affected to the formal parameter `mean` of the function `rnorm()`) which was set upon creation to the value $\mu = 7$ and which does not vary between calls of the function;
- the parameter **μ is thus an intrinsic characteristic** of each of the variables X_i .

Note

Since \bar{X}_4 is a random variable, it has a distribution, like any random variable. Mathematical theory states that $\bar{X}_4 \sim \mathcal{N}(\mu = 7, \sigma^2 = 1/4)$. If we only care about the behaviour of the distribution of `Xbar4()`, we can define it directly with

```
> Xbar4 <- function() rnorm(1, mean = 7, sd = sqrt(1/4))
```

SECTION 12.3 –

Law of Large Numbers and Central Limit Theorem

Let $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$ be the mean random variable made of n i.i.d. random variables X_i , each following the same distribution \mathcal{L} (which is not necessarily known) with expectation $\mathbb{E}(X_i) = \mu$ and variance $\text{Var}(X_i) = \sigma^2 < \infty$.

12.3.1 Law of Large Numbers

The **law of large numbers** states that when n goes to infinity, the mean random variable \bar{X}_n goes (technical term: converges in probability) to $\mathbb{E}(X_1)$, i.e. the theoretical mean μ . This is written $\bar{X}_n \xrightarrow{P} \mathbb{E}(X_1)$. This can easily be checked in R, for example when $\mathcal{L} = \mathcal{U}(0, 1)$:

```
> mean(runif(1))
[1] 0.2501903
> mean(runif(10))
[1] 0.3372082
```

```
> mean(runif(100))
[1] 0.5185887
> mean(runif(1000))
[1] 0.4885624
> mean(runif(10000))
[1] 0.5062192
> mean(runif(100000))
[1] 0.4998102
> mean(runif(1000000))
[1] 0.5000058
```

We notice that when the size of the sample n increases, these numbers get closer to the theoretical mean $\mu = 0.5$ of the n i.i.d. random variables $\mathcal{U}(0, 1)$.

Note

This will be very useful to approximate (technical term: **estimate**) the unknown parameters of a distribution. We shall return to this point in Sect. 12.4.



12.3.2 Central Limit Theorem

We can also look at the random variable $Y_n = \sqrt{n} \left(\frac{\bar{X}_n - \mu}{\sigma} \right)$ which is used as a pivot when constructing some confidence intervals and hypothesis tests. As all random variables, Y_n has a distribution. In some cases, mathematics allow us to calculate it explicitly. In general, this distribution depends on the size n of the sample. One might wonder how this distribution changes as n increases. This behaviour is called the convergence in distribution of a random variable. The **central limit theorem** states that Y_n converges in distribution to a random variable with distribution $\mathcal{N}(0, 1)$. In other words, generating observations from Y_∞ is equivalent (as far as one is only concerned with the distribution of the values of Y_∞) to generating observations from a random variable with distribution $\mathcal{N}(0, 1)$.

This mode of convergence and other classical modes of convergence (convergence in probability, almost sure, and in r -th order mean) are very well explained in [24].

The package `ConvergenceConcepts`, described in [23], is a tool to visualize this evolution graphically. For example, try the following instructions (Fig. 12.2):

```
require("ConvergenceConcepts")
investigate() # Choose Example 3.
```

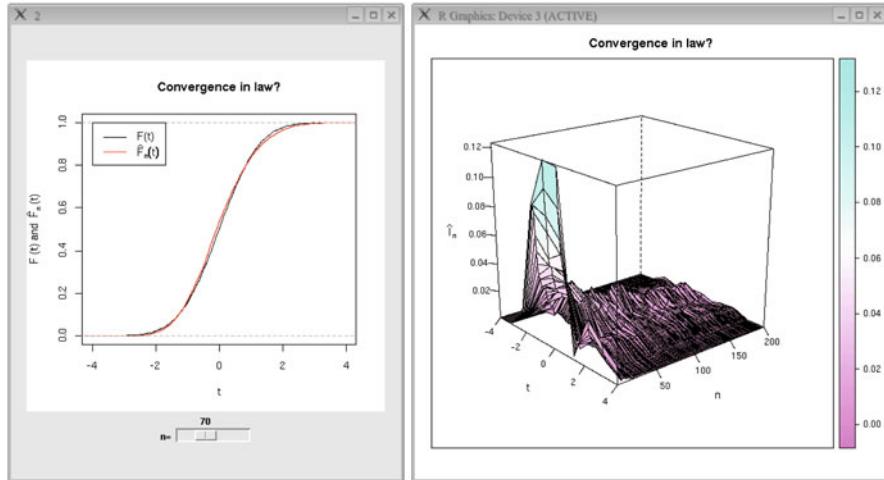


Fig. 12.2: Convergence in distribution in action on an example with simulated data. *Left:* the cumulative distribution function of a $\mathcal{N}(0, 1)$ is plotted in black; the empirical cumulative distribution function \hat{F}_{Y_n} (see Sect. 12.4.2) of the sample Y_n ($n = 70$) based on $M = 5,000$ realizations is plotted in red. *Right:* 3D plot of $|\hat{F}_{Y_n}(t) - F(t)|$ as a function of n and t

SECTION 12.4

Inferential Statistics

In the previous section, we saw that the law of large numbers can be used to approximate the fixed parameter (call it θ) of a distribution, thanks to the realization of a random variable (e.g. the parameter μ , thanks to a realization of \bar{X}_n). This will of course be very useful when the parameter is unknown. The “approximating” random variable is then called an **estimator** of the parameter. Realizations of this estimator are called **estimates** (of the unknown parameter θ). To perform **inference** is to estimate unknown parameters based on a sample of random variables which follow a distribution which depends on these unknown parameters.

12.4.1 Point Estimate of Parameters

What is an estimator? Since we are proposing a plausible value for the unknown parameter of a distribution, based on a sample generated from this distribution, we

can consider this plausible value as a function of the sample. If θ is the unknown value of the parameter to guess (estimate), we note our proposition $\hat{\theta}(x_1, \dots, x_n)$, thus specifying explicitly that it depends on the values of our sample. We call it an estimate of θ .

Then $\hat{\theta}(x_1, \dots, x_n)$ is the estimator of θ . This estimator is a random variable, a function of the random variables which generated the sample, and was built so that it is close of θ in some sense.

There are several techniques to propose an estimator of a parameter θ . For example, recall that the law of large numbers implies that we can propose $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$ as an estimator of the theoretical expectation $\mu = \mathbb{E}(X_1)$. Note that the sample (X_1, \dots, X_n) was generated from a distribution which takes precisely this unknown value μ as one of its parameters.

```
> theta <- 7 # Value assumed unknown.
> mean(rnorm(10000,mean=theta)) # Use the observed sample
(x1,...,x10000).
[1] 7.019249
```

Following the same idea based on the law of large numbers, we can propose $\bar{x}_n^2 - (\bar{X}_n)^2 = \frac{1}{n} \sum_{i=1}^n x_i^2 - \left(\frac{1}{n} \sum_{i=1}^n X_i\right)^2$ as an estimator of the (assumed unknown) parameter $\sigma^2 = \text{Var}(X_1) = \mathbb{E}(X_1^2) - [\mathbb{E}(X_1)]^2$. To estimate σ^2 , we thus only used an observed sample of random variables (which follow a distribution depending on the unknown parameter σ).

Note

This estimator is biased (see Sect. 12.4.4). An unbiased estimator of σ^2 is $\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{X}_n)^2$.



```
> theta <- 2 # Var(X_1), assumed unknown.
> vecx <- rnorm(10000, sd=sqrt(theta))
> mean(vecx^2) - (mean(vecx))^2 # We only used the observed sample.
[1] 1.976646
```

Tip

Note that this approach can be used to compute integrals. Indeed, for example,

$$\int_a^b g(x)dx = \int_{\mathbb{R}} g(x)\mathbb{1}_{[a,b]}(x)dx = \mathbb{E}[g(X)],$$

where X is a random variable distributed along the uniform distribution $\mathcal{U}(a, b)$ over the interval $[a, b]$.



It thus suffices to generate a sample (x_1, \dots, x_n) , with large size n , from random variables following a distribution $\mathcal{U}(a, b)$, to estimate $\mathbb{E}[g(\mathbf{x})]$ with

$$\frac{1}{n} \sum_{i=1}^n g(x_i).$$

This is called integral computation by **Monte Carlo simulation**.

12.4.2 Empirical Cumulative Distribution Function

A random variable following a Bernoulli distribution with parameter p is defined as a generator to create only 1's and 0's, with probability p of creating a 1 and $1 - p$ of creating a 0. Given a sample (x_1, \dots, x_n) of n realizations of random variables following this distribution, we can then compute the proportion \hat{p} of values equal to 1 amongst all the values of the sample. This proportion is also equal to the mean $\bar{x}_n = \frac{1}{n} \sum_{i=1}^n x_i$.

By the law of large numbers, as n gets larger, this mean gets closer to the expectation of X , i.e. $\mathbb{E}(X) = 1 \times P[X = 1] + 0 \times P[X = 0] = P[X = 1] = p$. This is called the **frequentist approach of probability**, which stipulates that the probability of an event is defined as the limit of the frequency that this event occurred. We note \hat{p} the estimator of the proportion p .

Let X be a random variable and x a real value (here, x is not a realization of X). We can then create a new random variable $\mathbb{1}_{[X \leq x]}$ which takes the values 1 or 0 depending on whether X takes values lower or greater than x . Following the same idea, and using a sample $\mathbf{X}_n = (X_1, \dots, X_n)^\top$, we can create the random variable $\hat{F}_n(x) := \hat{F}_{\mathbf{X}_n}(x) := \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{[X_i \leq x]}$. The law of large numbers states that

$$\frac{1}{n} \sum_{i=1}^n \mathbb{1}_{[X_i \leq x]} \xrightarrow{P} \mathbb{E}(\mathbb{1}_{[X_1 \leq x]}).$$

It can be shown theoretically that $\mathbb{E}(\mathbb{1}_{[X_1 \leq x]}) = P[X_1 \leq x] = F_{X_1}(x)$. Thus,

$$\hat{F}_{\mathbf{X}_n}(x) \xrightarrow{P} F_{X_1}(x).$$

The random variable $\hat{F}_{\mathbf{X}_n}(x)$, seen as a function of x , is called the **empirical cumulative distribution function** of the sample $\mathbf{X}_n = (X_1, \dots, X_n)$. We shall return to this function when we introduce the bootstrap method in Sect. 12.6.

The R function `ecdf()` can be used to create simply the empirical cumulative function:

```
> X <- function() rnorm(1)
> vecXn <- function(n) replicate(n,X())
# Sample of r.v.s
# following the
# distribution  $\mathcal{N}(0, 1)$ .
> Fnhat <- function(n,x,X) ecdf(X(n))(x) # Creating function
# (r.v.)  $\hat{F}_n$ .
> Fnhat(n=10,x=0,X=vecXn) # First call of  $\hat{F}_n(x)$  with  $x = 0$ .
[1] 0.6
> Fnhat(n=1000,x=0,X=vecXn) # Second call of  $\hat{F}_n(x)$  with  $x = 0$ .
[1] 0.509
```

12.4.3 Maximum Likelihood Estimation

The problem is as follows. We observe in nature some phenomenon and gather data related to this phenomenon. We thus have an observed sample of data (x_1, \dots, x_n) of size n . We can then assume that these data were created by “Mother Nature”, who generated them with a random variable following a distribution, say $\mathcal{N}(\theta^*, 1)$, for some parameter θ^* whose exact value is unknown, but which belongs to the set $\{0, 1, 2, \dots, 9\}$.

The aim of the **method of maximum likelihood** is to guess (estimate) the value of θ^* , based only on the observed sample, which is the only piece of **information** available about the data **generation process** (apart from the hypothesis on the distribution of the random variable which generated the data). We are trying to find out which is the most plausible value for θ^* (in $\{0, 1, \dots, 9\}$): the one most likely to have led to generating the data we observed.

We shall use a **computer simulation**, which will help us better understand the mechanism of the maximum likelihood method.

To this end, let us pretend for a moment that we are “Mother Nature” (or the “Great Architect”) and choose a value θ^* in the set $\{0, 1, 2, \dots, 9\}$, called the true value of the parameter θ . To allow us (“Mother Nature”) to choose a value while allowing us (the “Statistician”) to leave it unknown temporarily, we use the function `runif()`:

```
> theta.point <- as.integer(runif(1)*10)
> # For now, do not display the value of theta.point.
```

Now, still as “Mother Nature”, we simulate a sample (x_1, \dots, x_n) of size n from a distribution $\mathcal{N}(\theta^*, 1)$.

```
> n <- 1000
> x1...xn <- rnorm(n,mean=theta.point)
```


Warning

As we mentioned in part II, it is allowable to include dots (.) in the name of a variable. Thus `x1...xn` represents the name of an R variable.

We can now shed the clothes of “Mother Nature” and return to our simple statistician’s envelope. We now wish to estimate the unknown θ^* with a numerical value, which we note $\hat{\theta}(x_1, \dots, x_n)$. To this end, we shall use the R optimization function `nlinmb()` to create a function $\hat{\theta}(X_1, \dots, X_n)$ which builds this estimate. It will be built as the value of θ which maximizes the likelihood $\mathcal{L}(\theta; X_1, \dots, X_n)$ of the sample or, equivalently, which minimizes $-\text{Log}\mathcal{L}(\theta; X_1, \dots, X_n)$. This function is called the **maximum likelihood estimator** of θ^* . Note that the likelihood (evaluated at θ) is in a sense a measure of the plausibility of observing the sample if we assume that $\theta^* = \theta$.

We define

```
> theta.hat <- function(X1...Xn) {
+   start <- 0.5
+   nlinmb(start,minus.log.likelihood,X1...Xn=X1...Xn)$par
+ }
```

where the function `minus.log.likelihood()` is defined as

```
> minus.log.likelihood <- function(theta,X1...Xn) {
+   res <- -sum(log(dnorm(X1...Xn,theta)))
+   return(res)
+ }
```

Now, we can use our estimator on the observations x_1, \dots, x_n given by “Mother Nature” to get our estimate.

```
> theta.hat(x1...xn)
[1] 9.024612
```

Since we assumed that the value θ^* belongs to the set $\{0, 1, \dots, 9\}$, we can propose as an estimate of θ^* the value $\hat{\theta}^* = 9$.

We can now return to the role of “Mother Nature” and check whether the estimate we got is close to the true unknown value θ^* which we had temporarily hidden.

```
> theta.point
[1] 9
```

Note that we had assumed here that θ^* lives in a discrete set (finite or countable), which allowed us to find its exact value with our estimate. This will not be possible with continuous values of parameters to estimate.

Warning

One of the main advantages of introducing statistics with simulations is that we can be simultaneously (or at least alternately) on both sides of the fence:



“Mother Nature” | Statistician

12.4.4 Sampling Variation and Properties of an Estimator

- **Sampling variation**

In the practical section, we shall see how to create a tool which simulates the throw of a die. We use it now to throw 20 virtual dice:

```
> n <- 20
> res <- throw.die(n)
> res
[1] 2 3 4 6 2 6 6 4 4 1 2 2 5 3 5 3 5 6 3 5
```

We saw in Sect. 12.4.2 that we can estimate the probability p of getting a 4 by the proportion of times we got a 4 in the above sample: $\hat{p} = 0.15$.

Since a die has six sides, the expected value is $1/6 \approx 0.1667$. This is not quite the value we got. Let us throw 20 other virtual dice to see what is going on.

```
> res <- throw.die(n)
> res
[1] 2 5 3 2 4 4 1 2 4 4 4 4 6 5 1 5 6 2
```

This time, our estimate of the probability p of getting a 4 is $\hat{p} = 0.4$. The estimate has changed with this new sample. This phenomenon is called **sampling variation**. When we try to estimate an unknown parameter θ , the estimates vary as a function of the samples. Each new observed sample leads to a different estimate of θ .

If the size of the sample increases (e.g. $n = 10,000$), there is much less variation between two throws of 10,000 dice.

```
> n <- 10000
> res <- throw.die(n)
> sum(res==4)/n
[1] 0.1725
> res <- throw.die(n)
> sum(res==4)/n
[1] 0.1678
```

• Properties of an estimator

How can we know whether an estimator $\hat{\theta}(X_1, \dots, X_n)$ is accurate enough to estimate an unknown parameter θ ? We can define two criteria for an estimator:

- its **bias** (when estimating θ) $\mathbb{E}[\hat{\theta}(X_1, \dots, X_n); \theta] = \mathbb{E}[\hat{\theta}(X_1, \dots, X_n)] - \theta$, which measures whether the estimator is correct “on average”;
- its **variance** $\text{Var}[\hat{\theta}(X_1, \dots, X_n)]$ which measures the variability of the estimator

Note that the bias and variance of $\hat{\theta}(X_1, \dots, X_n)$ depend (theoretically) on n .

Now, suppose we have a generator which simulates the behaviour of random variables X_1, \dots, X_n , i.e. we are able to create a large number of samples (X_1, \dots, X_n) (say $M = 10,000$ or even more, depending on the context). It is then possible, with Monte Carlo simulation, to estimate the bias and variance of an estimator. Indeed, for each observed sample $(x_{1,i}, \dots, x_{n,i})$, we can compute the corresponding estimate $\hat{\theta}(x_{1,i}, \dots, x_{n,i})$. We then have M values $\hat{\theta}_1, \dots, \hat{\theta}_M$ which can be used to estimate $\mathbb{E}[\hat{\theta}(X_1, \dots, X_n); \theta]$ with $\bar{\hat{\theta}} = \frac{1}{M} \sum_{i=1}^M \hat{\theta}_i$ and the variance $\text{Var}[\hat{\theta}(X_1, \dots, X_n)]$ with $\frac{1}{M} \sum_{i=1}^M (\hat{\theta}_i - \bar{\hat{\theta}})^2$.

In the next example, we use Monte Carlo simulation to estimate the bias and variance of the estimator \hat{p} (frequency of 4's out of n dice throws) of the known parameter $p = \theta = 1/6$. Note that p is the parameter of the random variable X which represents whether we get a 4 when throwing a die. The distribution of X is a Bernoulli with parameter $p = 1/6$.

```
> n <- 20
> M <- 100000
> vec.theta.hat <- replicate(M, {res <- throw.die(n)
+ theta.hat <- sum(res==4)/n})
> mean(vec.theta.hat)-1/6 # estimate of the bias.
[1] -0.00006716667
> var(vec.theta.hat)      # estimate of the variance.
[1] 0.006969351
```

For this example, it can be shown that the random variable $n\hat{\theta}(X_1, \dots, X_n)$ follows a binomial distribution $\text{Bin}(n, p)$, with expectation np and variance $np(1-p)$ (here, $p = 1/6$). Thus the estimator $\hat{\theta}(X_1, \dots, X_n)$ is an unbiased estimator of θ and its variance is $\frac{p(1-p)}{n}$.

We can check this numerically:

```
> p <- 1/6
> p*(1-p)/n
[1] 0.006944444
```

See also

The bootstrap technique, which we introduce in Sect. 12.6, can be used to approximate the bias and variance of a given estimator, based on a single sample (x_1, \dots, x_n) (which is often all we have in real life), rather than on M samples as we did in Monte Carlo simulation, where a data generator is available.

**SECTION 12.5****A Few Techniques to Draw from a Distribution**

Why performing simulations?

- to “check”, using the computer, a mathematical result already known;
- to “demonstrate”, using the computer, a result we cannot prove theoretically;
- this can guide us in the demonstration of a difficult mathematical result;
- it is often required when trying to publish results in scientific journals;
- this allows better reasoning as a statistician, as we have already seen in this chapter.

How should one perform efficient simulations? One must:

- (1) properly identify the (heart of) the problem;
- (2) write an outline of an algorithm to solve the problem;
- (3) translate this algorithm into a program written in an interpreted language like **R**;
- (4) test this program to ensure it works properly;
- (5) translate (possibly) the program using a lower level language (compiled) as **C/C++** or **Fortran**.

Any simulation needs one to be able to simulate random variables from a given law. In the previous sections, we saw how to simulate from a few distributions in **R** (`rnorm()`, `runif()`). When the distribution to simulate from is not implemented in **R**, you can use one of the methods introduced in this section.

12.5.1 Simulating from Another Distribution

There sometimes exists a simple formula expressing the random variable X with distribution \mathcal{L} , from which we wish to sample observations, as a function of one or several random variables with a standard distribution. It is then easy to build a generator with distribution \mathcal{L} , thanks to this formula. The very simple following example illustrates this point. For example, recall that a random variable following

a χ_1^2 distribution can be obtained by taking the square of a standard normal random variable.

```
> rchi2.1 <- function() rnorm(1)^2 # X ~ N(0, 1) ⇒ X^2 ~ χ_1^2.
```

Do it yourself



Generate observations from a $TU(2)$ distribution (see the definition of this distribution in Sect. 12.7, page 409).

12.5.2 Inverse Transform Method

Suppose we know the inverse cumulative distribution function F_X^{-1} of the random variable X , and we wish to draw observations following the same distribution as X . This is very easy to do with a generator U with distribution $\mathcal{U}(0, 1)$, thanks to the following formula:

$$\tilde{X} = F_X^{-1}(U).$$

Indeed, the cumulative distribution function of the random variable \tilde{X} happens to be $F_{\tilde{X}}$. This property is known as *inverse transform method* and was discovered by R.A. Fisher ([16]).

Do it yourself



Recall that the cumulative distribution function of an exponential random variable X with distribution $\mathcal{E}(\lambda)$ is

$$F_X(x; \lambda) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

Calculate F_X^{-1} and then use the function `runif()` to generate observations from the distribution $\mathcal{E}(2)$.

12.5.3 Rejection Sampling

Suppose we know the density function f_X of the random variable X , but not its inverse cumulative distribution. We wish to simulate observations following the

same distribution as X . Rejection sampling consists in generating data following a distribution with density g close to f (in the sense that $f_X(x) \leq c \times g(x)$ for some $c > 0$) and then discarding some proportion of these data, so as to get data which follow the desired distribution.

The algorithm is as follows:

- (1) generate a data point y using a random variable Y with density g ;
- (2) generate u from a random variable U with density $\mathcal{U}(0, 1)$;
- (3) if $c \times g(y) \times u \leq f(y)$, then keep y as a generated data point; else discard it and return to the start of the algorithm.

The values output by this algorithm can be used as if they were generated by a random variable with density f_X . The best value for the constant c to minimize the number of rejections is $c = \sup_x \frac{f_X(x)}{g(x)}$.

Do it yourself



Use the rejection method to generate observations from the distribution $\mathcal{N}(0, 1)$, using as reference function the density of an exponential distribution with parameter $\lambda = 1$. Take the value $c = \sqrt{\frac{e^1}{2\pi}}$, and use the function `rbinom()` to assign a positive or negative sign to the values given by the rejection algorithm.

12.5.4 Simulation of Discrete Random Variables

Suppose we wish to simulate a sample from a discrete random variable X which satisfies $P(X = x_i) = p_i$ for all i in \mathbb{N} (or a subset of \mathbb{N}). Define U as a random variable with distribution $\mathcal{U}(0, 1)$, which can be sampled from with the function `runif()`, and use the following algorithm:

$$\begin{cases} X = x_0 & \text{if } 0 < U \leq p_0; \\ X = x_i & \text{if } \sum_{j=0}^{i-1} p_j < U \leq \sum_{j=0}^i p_j. \end{cases}$$

Do it yourself



Use this algorithm to generate observations from the discrete uniform distribution over the set $\{0, 1, 2, 3, 4, 5\}$.

SECTION 12.6

Bootstrap

Resampling methods, also known as bootstrap methods, consist in using the information available in a sample of observed values (x_1, \dots, x_n) to approximate the distribution of the i.i.d. random variables X_1, \dots, X_n which generated this sample. We saw in Sect. 12.2.3.1 that the distribution of a random variable can be described by the cumulative distribution function F_X , and we saw in Sect. 12.4.2 how the cumulative distribution function can be approximated with the empirical cumulative distribution function \hat{F}_{X_n} . The idea behind bootstrap is to generate several observed data sets $\mathbf{x}_1^* = (x_{1,1}^*, \dots, x_{n,1}^*), \dots, \mathbf{x}_B^* = (x_{1,B}^*, \dots, x_{n,B}^*)$ following the distribution described by \hat{F}_{X_n} (which we know), as a proxy for F_X (which is unknown in practice). We then consider that these new data which can be generated have the same properties as observations we would get if we could use the generator based on F_X (which we cannot in practice). We can then use Monte Carlo techniques, as introduced in Sect. 12.4.1, to estimate for example the bias and the variance of the estimator $\hat{\theta}(X_1, \dots, X_n)$ of some unknown parameter θ , with the formulae $\frac{1}{B} \sum_{b=1}^B \hat{\theta}(x_{1,b}^*, \dots, x_{n,b}^*) - \hat{\theta}(x_1, \dots, x_n)$ and $\frac{1}{B} \sum_{b=1}^B \left(\hat{\theta}(x_{1,b}^*, \dots, x_{n,b}^*) - \frac{1}{B} \sum_{j=1}^B \hat{\theta}(x_{1,i}^*, \dots, x_{n,i}^*) \right)^2$. One question still needs to be answered: how do we generate observations from \hat{F}_{X_n} ? This is in fact very simple: all we need to do is draw with replacement n observations from the original sample. The bootstrap procedure consists in generating B such samples \mathbf{x}_b^* and use them as illustrated in the above example. The R function `sample()` is used for the draw with replacement operation.

We return to the example from Sect. 12.4.4, where we tried to estimate the bias and variance of the estimator \hat{p} (frequency of the number 4 for n throws of a die) of the parameter $p = \theta$ of the random variable X which represents whether we get a 4 when throwing a die.

```
> n <- 20 ; xvec <- throw.die(n) ; sum(xvec==4)/n
[1] 0.15
```

Draw with replacement a sample of length n from xvec:

```
> sample(xvec,n,replace=TRUE)
[1] 6 6 4 5 6 2 5 2 5 4 6 6 5 5 6 2 1 4
> B <- 10000
> vec.theta.star <- replicate(B,sum(
+           sample(xvec,n,replace=TRUE)==4)/n)
> mean(vec.theta.star) - sum(xvec==4)/n
[1] 0.00009
> ((B-1)/B)*var(vec.theta.star) ; sd(vec.theta.star)
[1] 0.006377992
[1] 0.07986632
```

In this very simple case, theory states that the bias is zero and the variance is $p(1-p)/n = 0.00694$.

Tip

Note that the package `boot()` facilitates bootstrap:
`boot(xvec, function(x,w) sum(x[w]==4)/n, B)`



SECTION 12.7

Standard and Less Standard Distributions**12.7.1 Standard Distributions**

Standard probability distributions are implemented in R. In Tables 12.1 and 12.2, we give the functions to compute the density (or mass probability function), the cumulative distribution function and the quantile function of these distributions. We also give the instruction to generate pseudo-random numbers from these distributions.

Table 12.1: Standard discrete distributions. R functions for the mass function (d--), cumulative distribution function (p--) and quantile function (q--). Instruction to generate (r--) pseudo-random numbers from these distributions

Discrete distributions	R functions	Expected value Variance	Probability mass functions $P(X = x)$
Binomial(m, α)	<code>dbinom(x, size=m, prob=alpha)</code>	$m\alpha$	$\binom{m}{x} \alpha^x (1-\alpha)^{m-x}$
	<code>pbinom(q, size=m, prob=alpha)</code>		
	<code>qbinom(p, size=m, prob=alpha)</code>	$m\alpha(1-\alpha)$	
	<code>rbinom(n, size=m, prob=alpha)</code>		
Poisson(λ)	<code>dpois(x, lambda=lambda)</code>	λ	$e^{-\lambda} \frac{\lambda^x}{x!}$
	<code>ppois(q, lambda=lambda)</code>		
	<code>qpois(p, lambda=lambda)</code>	λ	
	<code>rpois(n, lambda=lambda)</code>		
Geometric(α)	<code>dgeom(x, prob=alpha)</code>	$\frac{1}{\alpha}$	$(1-\alpha)^{x-1}\alpha$
	<code>pgeom(q, prob=alpha)</code>		
	<code>qgeom(p, prob=alpha)</code>	$\frac{1-\alpha}{\alpha^2}$	
	<code>rgeom(n, prob=alpha)</code>		
Hyper-geometric(m, n, k)	<code>dhyper(x, m=m, n=n, k=k)</code>	$\frac{nm}{N}$ (with $N = n + m$)	$\binom{m}{x} \binom{n}{k-x} / \binom{m+n}{k}$
	<code>phyper(q, m=m, n=n, k=k)</code>		
	<code>qhyper(p, m=m, n=n, k=k)</code>		
	<code>rhyper(nn, m=m, n=n, k=k)</code>		
Negative binomial(m, α)	<code>dnbinom(x, size=m, prob=alpha)</code>	$m \frac{1-\alpha}{\alpha}$	$\binom{x+m-1}{m-1} \alpha^m (1-\alpha)^x$
	<code>pnbinom(q, size=m, prob=alpha)</code>		
	<code>qnbinom(p, size=m, prob=alpha)</code>	$m \frac{1-\alpha}{\alpha^2}$	
	<code>rnbinom(n, size=m, prob=alpha)</code>		
Discrete uniform{1, ..., m}	<code>(x %in% 1:m)/m</code>	$\frac{m+1}{2}$	$\frac{1}{m} \mathbb{1}_{\{1, \dots, m\}}(x)$
	<code>sum(1:m<=q)/m</code>		
	<code>match(1, 1:m/m>p)</code>		
	<code>sample(x=1:m, size=n, TRUE)</code>	$\frac{m^2-1}{12}$	

Table 12.2: Standard continuous distributions. **R** functions for the density function (**d--**), cumulative distribution function (**p--**) and quantile function (**q--**). Instruction to generate (**r--**) pseudo-random numbers from these distributions (notations: $B(\cdot)$: beta function; $I(\cdot)$: modified Bessel function; $\Gamma(\cdot)$: gamma function; $P(\cdot; \lambda)$: mass function of a Poisson(λ); $I'_x(\cdot, \cdot)$: derivative of an incomplete beta function; $\text{sech}(x) = \frac{2}{e^x + e^{-x}}$)

Continuous distributions	R functions	Expected value	Variance	Density
Normal(μ, σ^2)	dnorm(x, mean=μ, sd=σ) pnorm(q, mean=μ, sd=σ) qnorm(p, mean=μ, sd=σ) rnorm(n, mean=μ, sd=σ)	μ σ^2		$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Student(v, μ)	dt(x, df=v, ncp=μ) pt(q, df=v, ncp=μ) qt(p, df=v, ncp=μ) rt(n, df=v, ncp=μ)	$\mu \sqrt{\frac{v}{2}} \frac{\Gamma((v-1)/2)}{\Gamma(v/2)}$ ($v > 1$) $\frac{v(1+\mu^2)}{v-2} - \frac{\mu^2 v}{2} \times \left(\frac{\Gamma((v-1)/2)}{\Gamma(v/2)} \right)$, ($v > 2$)		$\frac{v^{v/2} e^{-v/2} \mu^2 / 2(x^2+v)}{\sqrt{\pi} \Gamma(v/2) 2^{(v-1)/2} (x^2+v)^{(v+1)/2}}$ $\times \int_0^\infty t^v e^{-\frac{\mu x}{\sqrt{x^2+v}}} -\frac{t}{2} dt$
Chi-squared(k, λ)	dchi.sq(x, df=k, ncp=λ) pchi.sq(q, df=k, ncp=λ) qchi.sq(p, df=k, ncp=λ) rchisq(n, df=k, ncp=λ)	$k + \lambda$		$\frac{1}{2} e^{-(x+\lambda)/2} \left(\frac{x}{\lambda} \right)^k / 4^{-1/2}$ $\times I_{k/2-1}(\sqrt{\lambda}x)$
Fisher(v_1, v_2, λ)	df(x, df1=v ₁ , df2=v ₂ , ncp=λ) pf(q, df1=v ₁ , df2=v ₂ , ncp=λ) qf(p, df1=v ₁ , df2=v ₂ , ncp=λ) rf(n, df1=v ₁ , df2=v ₂ , ncp=λ)	$2(k + 2\lambda)$		$\sum_{k=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^k}{B(\frac{v_1}{2}, \frac{v_2}{2} + k) k!} \left(\frac{v_1}{v_2} \right)^{\frac{v_1}{v_2} + k}$ $\times \left(\frac{v_2}{v_2 + v_1 x} \right)^{\frac{v_1}{2} + k} x^{\frac{v_1}{2} - 1 + k}$
Exponential(λ)	dexp(x, rate=λ) pexp(q, rate=λ) qexp(p, rate=λ) rexp(n, rate=λ)			$\lambda e^{-\lambda x} \mathbb{1}\{x \geq 0\}$

<code>dunif(x, min=a, max=b)</code>	$\frac{a+b}{b-a} \mathbb{1}\{a \leq x \leq b\}$
<code>Uniform(a, b)</code>	$\frac{1}{b-a}$
<code>puniif(q, min=a, max=b)</code>	$\frac{(b-a)^2}{12}$
<code>quniif(p, min=a, max=b)</code>	$\approx 1 - \frac{\beta}{C} \left(1 + \frac{\lambda}{2C^2} \right)$ with $C = \alpha + \beta + \frac{\lambda}{2}$
<code>runiif(n, min=a, max=b)</code>	$\sum_{i=0}^{\infty} P(i; \frac{\lambda}{2}) I_x'(\alpha + i, \beta)$
<code>dbeta(x, shape1=α, shape2=β, ncp=λ)</code>	$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$ if $\lambda = 0$
<code>pbeta(q, shape1=α, shape2=β, ncp=λ)</code>	Undefined
<code>qbeta(p, shape1=α, shape2=β, ncp=λ)</code>	$\frac{1}{\pi} \left[\frac{\gamma}{(\alpha-x_0)^2 + \gamma^2} \right]$
<code>rbeta(n, shape1=α, shape2=β, ncp=λ)</code>	Undefined
<code>dcauchy(x, location=x₀, scale=γ)</code>	Defined
<code>pcauchy(q, location=x₀, scale=γ)</code>	$\frac{1}{4s} \operatorname{sech}^2 \left(\frac{x-\mu}{2s} \right)$
<code>qcauchy(p, location=x₀, scale=γ)</code>	Defined
<code>rcauchy(n, location=x₀, scale=γ)</code>	Defined
<code>dlogis(x, location=μ, scale=s)</code>	$e^{\mu + \sigma^2 / 2}$
<code>plogis(q, location=μ, scale=s)</code>	$e^{(\sigma^2 - 1)e^{2\mu + \sigma^2}}$
<code>qlogis(p, location=μ, scale=s)</code>	$\alpha\beta$
<code>rlogis(n, location=μ, scale=s)</code>	$\alpha\beta^2$
<code>dlnorm(x, meanlog=μ, sdlog=σ)</code>	$x^{\alpha-1} \frac{e^{-x/\beta}}{\beta^\alpha \Gamma(\alpha)} \mathbb{1}_{x>0}$
<code>plnorm(q, meanlog=μ, sdlog=σ)</code>	$\frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$
<code>qlnorm(p, meanlog=μ, sdlog=σ)</code>	$\lambda^k \left(\frac{x}{\lambda} \right)^{k-1} e^{-(x/\lambda)^k} \mathbb{1}_{x\geq 0}$
<code>rlnorm(n, meanlog=μ, sdlog=σ)</code>	$\lambda^2 \Gamma \left(1 + \frac{1}{k} \right)$
<code>dgamma(x, shape=α, rate=β)</code>	$\lambda^2 \Gamma \left(1 + \frac{2}{k} - \mu^2 \right)$
<code>pgamma(q, shape=α, rate=β)</code>	$\mu + \beta$
<code>qgamma(p, shape=α, rate=β)</code>	$\frac{z e^{-z}}{\beta}$ with $z = e^{-\frac{x-\mu}{\beta}}$
<code>rgamma(n, shape=α, rate=β)</code>	$\frac{\pi^2}{6} \beta^2$
<code>dweibull(x, shape=λ, scale=k)</code>	Defined
<code>pweibull(q, shape=λ, scale=k)</code>	Defined
<code>qweibull(p, shape=λ, scale=k)</code>	Defined
<code>rweibull(n, shape=λ, scale=k)</code>	Defined
<code>dgumbel(x, loc=μ, scale=β)</code>	Defined
<code>pgumbel(q, loc=μ, scale=β)</code>	Defined
<code>qgumbel(p, loc=μ, scale=β)</code>	Defined
<code>rgumbel(n, loc=μ, scale=β)</code>	Defined

12.7.2 † Less Standard Distributions

In the next tables, we give formulae to generate a sample from a few less standard distributions.

Table 12.3 presents the following distributions: *Rademacher* Rad, *Irwin-Hall* Irw(n), *Kumaraswamy* Kum(a, b), *inverse normal* GI(μ, λ), *Lévy* Levy(c), *Log-logistic* Log-Logis(α, β), *Rayleigh* Ray(σ^2), *Rice* Rice(σ, v), *multinomial* $\mathcal{M}(n, p_1, \dots, p_k)$.

Table 12.3: Less standard distributions (notations: $B_{1/2} \sim \text{Bernoulli}(1/2)$, $Y_{1,b} \sim \text{Beta}(1, b)$, $Z \sim \mathcal{N}(0, 1)$, $U_k, U \sim \mathcal{U}(0, 1)$, $G(\alpha, \beta) \sim \text{Gamma}(\alpha, \beta)$, $L_{1/2}(x) = e^{x/2} [(1-x)I_0\left(\frac{-x}{2}\right) - xI_1\left(\frac{-x}{2}\right)]$, $I_\alpha(\cdot)$: modified Bessel functions, $B(\alpha, \beta)$: beta function)

Distribution	Density	Generation	Expected value Variance
Rad	$1/2$ if $k = \pm 1$	$2B_{1/2} - 1$	0 1
Irw(n)	$\frac{1}{2(n-1)!} \sum_{k=0}^n (-1)^k \binom{n}{k} \times (x-k)^{n-1} sgn(x-k)$	$\sum_{k=1}^n U_k$	$n/2$ $n/12$
Kum(a, b)	$abx^{a-1}(1-x^a)^{b-1}$	$X_{a,b} = Y_{1,b}^{1/a}$	$bB(1+1/a, b)$ $bB(1 + \frac{2}{a}, b) - b^2 B^2(1 + \frac{1}{a}, b)$
GI(μ, λ)	$\left[\frac{\lambda}{2\pi x^3}\right]^{1/2} \exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right)$	$X = \mu + \frac{\mu^2}{2\lambda} \left[Z^2 - \frac{ Z }{\mu} \sqrt{4\mu\lambda + \mu^2 Z^2} \right]$ $X \text{ if } U \leq \frac{\mu}{\mu+X}, \text{ else } \frac{\mu^2}{X}$	μ $\frac{\mu^3}{\lambda}$
Levy(c)	$\sqrt{\frac{c}{2\pi}} \frac{e^{-c/2x}}{x^{3/2}}$	$X = \frac{1}{G(1/2, c/2)}$	∞ ∞
Log-Logis(α, β)	$\frac{(\beta/\alpha)(x/\alpha)^\beta - 1}{[1 + (x/\alpha)^\beta]^2}$	$X = \exp(\text{Logistic}(\log(\alpha), \beta))$	$\alpha^2 \left(\frac{2b}{\sin 2b} - \frac{b^2}{\sin^2 b} \right)$ if $\beta > 1$ $\alpha^2 \left(\frac{2b}{\sin 2b} - \frac{b^2}{\sin^2 b} \right)$ if $\beta > 2$
Ray(σ^2)	$\frac{x}{\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right)$	$X = \sigma \sqrt{-2 \log(U)}$	$\frac{\sigma \sqrt{\frac{\pi}{2}}}{4 - \frac{\pi}{2} \sigma^2}$
Rice(σ, v)	$\frac{x}{\sigma^2} \exp\left(-\frac{(x^2+v^2)}{2\sigma^2}\right) I_0\left(\frac{xy}{\sigma^2}\right)$	$R = \sqrt{X^2 + Y^2}$ with $X \sim \mathcal{N}(1, \sigma^2)$ and $Y \sim \mathcal{N}(0, \sigma^2)$	$\frac{\sigma \sqrt{\pi/2} L_{1/2}(-v^2/2\sigma^2)}{2\sigma^2 + v^2 - \frac{\pi\sigma^2}{2} L_{1/2}^2\left(\frac{-v^2}{2\sigma^2}\right)}$
$\mathcal{M}(n, p_1, \dots, p_k)$	$\frac{n!}{x_1! \dots x_k!} p_1^{x_1} \dots p_k^{x_k}$ if $\sum_{i=1}^k x_i = n$	$Y_j = \arg \min_{j'=1}^k \left(\sum_{i=1}^{j'} p_i \geq U \right)$ $X = \sum_{j=1}^n Y_j$ (Y_j i.i.d.)	$\mathbb{E}(X_i) = p_i$ $\text{Var}(X_i) = np_i(1-p_i)$

Table 12.4 gives formulae to generate a sample from the following less standard distributions: *skew-normal* SN(ξ, ω^2, α), *Laplace* Lp(μ, b), *shifted exponential* SE(l, b), *generalized Pareto* GP(μ, σ, ξ), *generalized error distribution* GED(μ, σ, p), *Johnson SU* JSU(μ, σ, v, τ), *symmetrical Tukey* TU(l), *scale contaminated SC*(p, d), *location contaminated LC*(p, m), *Johnson SB* SB(g, d), *stable* S(a, b).

Table 12.4: Less standard distributions (notations: $U \sim \mathcal{U}(0,1)$, $Z \sim \mathcal{N}(0,1)$, $Z_\mu \sim \mathcal{N}(\mu,1)$, $V_d \sim \mathcal{N}(0,d)$, $E \sim \mathcal{E}(1)$, $B_p \sim \text{Bernoulli}(p)$, $G_p \sim \text{Gamma}(1/p, p)$, $F_\lambda^{-1}(p) = p^\lambda - (1-p)^\lambda$, $\Theta = \pi(U - \frac{1}{2})$, $\Theta_\beta = \frac{\pi}{2} + \beta\Theta$)

Distribution	Density	Generation	Expected value
$SN(\xi, \omega^2, \alpha)$	$\frac{2}{\omega\sqrt{2\pi}} e^{-\frac{(\xi-\mu)^2}{2\omega^2}} \int_{-\infty}^{\alpha(\frac{\xi-\mu}{\omega})} \frac{e^{-t^2/2}}{\sqrt{2\pi}} dt$	$X = \xi + \omega Y$ Y = W if $Z_0 \geq 0$; -W else $W = \delta Z_0 + \sqrt{1-\delta^2} V_1$ $\delta = \alpha/\sqrt{1+\alpha^2}$	$\xi + \omega \sqrt{2/\pi}\delta$ $\omega^2(1-2\delta^2/\pi)$
$LP(\mu, b)$	$\frac{1}{2b} \exp\left(-\frac{ x-\mu }{b}\right)$	$X = \mu - b \cdot \text{sgn}(U - \frac{1}{2}) \ln(1 - 2 U - \frac{1}{2})$	$\frac{\mu}{2b^2}$
$SE(l, b)$	$b \exp\{-l(x-l)b\}, x \geq l$	$X = \frac{-\ln U}{b} + l$	$l + \frac{1}{b^2}$
$GP(\mu, \sigma, \xi)$	$\frac{1}{\sigma} \left(1 + \frac{\xi - (\mu - \sigma)}{\sigma}\right)^{(-\frac{1}{\xi} - 1)}$ $x \geq \mu \text{ if } \xi \geq 0$ $x \leq \mu - \frac{\sigma}{\xi} \text{ if } \xi < 0$	$X = \mu + \frac{\sigma(U^{-\xi} - 1)}{\xi}$	$\mu + \frac{\sigma}{1-\xi} \quad (\xi < 1)$ $\frac{\sigma^2}{(1-\xi)^2(1-2\xi)}, \quad (\xi < \frac{1}{2})$
$GED(\mu, \sigma, p)$	$\frac{1}{2p^{1/p} \Gamma(1+1/p)\sigma} e^{-\frac{1}{p\sigma^p} x-\mu ^p}$	$\mu + \sigma G_p^{-1} \text{sgn}(U - 1/2)$	μ
$JSD(\mu, \sigma, \tau)$	$\frac{1}{c\sigma} \frac{1}{\sqrt{\pi}^{2-1}} \frac{1}{\sqrt{2\pi}} e^{-r^2/2}$ $r = -v + \tau \sinh(z)$ $z = \frac{\lambda - \mu + c\sigma \sqrt{w} \text{erf}(z/2)}{c\sigma}$ $c = \sqrt{((w-1)(w \cosh(2\Omega) + 1)/2)}$ $w = e^{1/(\tau^2)}$ and $\Omega = -v/\tau$	$X = \mu + c\sigma \times \left[\sqrt{w} \sinh(\Omega) + \sinh\left(\frac{Z_0 + v}{\tau}\right) \right]$	μ^2
$TU(\lambda)$	$\frac{1}{F_\lambda(x)^\lambda - 1}$	$X = U^\lambda - (1-U)^\lambda$	$\frac{0}{2\lambda + 1} - \frac{2^{2\lambda + 1}}{2\lambda + 2}$
$SC(p, d)$	$\frac{1}{\sqrt{2\pi}} [f'_d e^{-\frac{x^2}{2d^2}} + (1-p)e^{-\frac{x^2}{2}}]$	$B_p V_d + (1-B_p)Z_0$	$pd^2 + 1 - p$
$LC(p, m)$	$\frac{1}{\sqrt{2\pi}} [pe^{-\frac{(x-m)^2}{2}} + (1-p)e^{-\frac{x^2}{2}}]$	$B_p Z_m + (1-B_p)Z_0$	$\frac{mp}{1-(mp)^2}$
$SB(g, d)$	$\frac{d}{\sqrt{2\pi}} \frac{1}{x(1-x)} e^{-\frac{1}{2}(g+d \log \frac{x}{1-x})^2}, d > 0$	$X = \left(1 + e^{-\frac{Z_0 - g}{d}}\right)^{-1}$	No explicit formula
$S(\alpha, \beta)$ with $-1 \leq \beta \leq 1$ $0 < \alpha \leq 2$		$\theta_0 = -\frac{\pi}{2}\beta(\alpha - 1 + \text{sgn}(1-\alpha))/\alpha$ $\hat{X}_1 = \sin\{\alpha(\Theta - \theta_0)\}$ $X_2 = \{\cos(\Theta)\}_{1/\alpha}^{1/\alpha}$ $X_3 = \left[\frac{\cos((\alpha-1)\Theta - \alpha\theta_0)}{E}\right]_{\alpha}^{-1}$ If $\alpha \neq 1$: $X = (\hat{X}_1/\sqrt{2})(X_3)$, else: $X = \Theta_\beta \tan \Theta - \beta \log\left(\frac{E_{\text{ext}}(\Theta)}{E_\beta}\right)$	Undefined
	No analytical form		

SECTION 12.8

Modelling a Phenomenon

Suppose we observe the following $n = 500$ values produced by some phenomenon.

```
> xvec
[1] 1 1 1 2 0 0 1 1 1 2 1 0 1 1 1 0 1 1 2 2 1 2 1 1 2 2 1 2 1 1 2 2 1 2
[30] 1 1 0 0 3 1 2 2 3 2 1 1 0 0 2 1 0 0 1 0 2 1 3 1 1 0 1 1 1
[59] 1 1 0 0 1 1 2 2 1 1 0 0 0 1 2 1 0 1 1 3 1 0 2 1 0 1 0 0 2
[88] 1 0 0 1 2 1 1 1 2 1 1 2 0 1 0 1 1 3 2 3 1 2 1 2 2 0 1 2 3
[117] 1 2 1 2 0 1 1 1 0 0 2 1 2 3 2 0 2 0 1 1 2 2 0 1 2 0 0 3 0
[146] 2 1 1 0 3 2 0 0 0 1 0 0 1 3 0 2 1 1 1 2 1 2 3 2 1 1 2 2 2
[175] 4 1 0 1 1 1 1 0 1 1 1 1 2 2 2 0 0 0 0 1 2 1 1 1 2 2 0 1 2
[204] 2 0 2 2 4 1 2 2 2 1 0 2 2 1 2 0 0 2 2 2 1 0 1 2 2 1 1 1
[233] 2 0 2 1 2 1 2 2 1 1 1 0 1 2 0 2 2 2 0 2 1 0 0 2 1 1 0 2 3
[262] 1 2 1 0 1 1 1 1 2 0 4 2 0 2 4 2 2 2 0 0 4 0 3 0 3 3 1 2 2
[291] 2 3 2 4 1 1 3 0 1 0 1 0 1 1 2 2 0 1 0 2 0 1 2 1 2 0 0 0 0
[320] 1 2 1 1 4 2 1 1 1 3 1 1 2 0 0 2 1 2 0 3 0 2 1 0 1 0 2 2
[349] 1 2 3 3 1 2 1 1 2 2 2 2 2 1 2 1 0 2 1 1 2 3 3 1 1 0 1 1 2
[378] 1 1 0 1 1 2 2 1 1 0 1 0 0 1 2 0 2 0 2 0 1 0 3 2 2 1 2 3 1
[407] 2 0 0 1 2 2 2 1 0 0 1 0 0 1 1 1 2 2 1 3 0 5 2 2 0 0 2 0 1
[436] 0 1 1 0 2 1 4 1 0 2 1 1 3 1 0 2 3 1 0 3 1 2 1 3 0 1 0 0 1
[465] 1 1 1 4 2 1 2 2 0 1 0 2 0 1 0 0 3 0 2 1 2 3 2 2 2 1 1 1 1
[494] 1 1 1 1 1 2 1
```

Given these data, a statistician will try to describe (mathematically) the generative process which led to these values. The observed values seem to arise in an unpredictable fashion, and it seems difficult to find a logical (and deterministic) sequence which would explain them. But can they be described?

Inductive **inferential statistics** can be used to go from observed facts about the sample to the probability distribution in the population. To this end, a statistician will for example first assume that each of these observations is a realization of a single random variable X and that these observations are produced independently of one another. Mathematically, this can be stated as “Let $\mathbf{x}_n = (x_1, \dots, x_n)$ be an (observed) sample of the random vector $\mathbf{X}_n = (X_1, \dots, X_n)$ made of $n = 500$ independently and identically distributed (i.i.d.) random variables”. The statistician is thus assuming that “Mother Nature” owns a random number generator X , known to her alone, and that the statistician obviously does not know. The statistician’s aim is to guess what the generator is, as far as possible. This process is called **statistical modelling**.

The statistician then searches within its toolbox of models and starts with the model which seems most simple (parsimony principle) and adequate. The random variables at play are discrete, so he has the following probabilist models at its disposal:

- the binomial distribution $\text{Bin}(m, \alpha)$;
- the Poisson distribution $\mathcal{P}(\lambda)$.

Given the observations (numbers between 0 and 4), the binomial distribution is most appropriate.

The statistician now needs to estimate the unknown parameters of this model, i.e. plausible values for the parameters of the chosen distribution (here, m and α). This is the parameter estimation phase, which was described in Sect. 12.4.1.

Note

As you may have guessed, the 500 data above were simulated with a computer. We let ourselves on the other side of the fence (the side of “Mother Nature”) to generate these data. We can now reveal that we used the following generator:

```
X <- function() rbinom(500,5,1/4)
```



One of the main advantages of computers is that we can be simultaneously on either side of the fence, to better understand the phenomena around us. This is of course not possible when working with real data.

The many distributions we have introduced in this chapter can be used to model other types of phenomena. The reader should be able to find a distribution amongst those we have listed to model a specific phenomenon.

Note

For example, the following remarks are worth noting:

- the Bernoulli distribution (`rbinom(n, 1, p)`) is used when a random experiment has only two possible outcomes: success, with probability p , and failure, with probability $1 - p$;
- the negative binomial(k, p) distribution (`rnbinom(n, k, p)`) models the number of observations until the k th success (included);
- the Poisson(λ) distribution (`rpois(n, lambda)`) is the distribution of a variable X which counts the number of realizations of a rare event, for example, per unit of time or per unit of area;
- the exponential(λ) distribution (`rexp(n, lambda)`) is used to model the time at which a system breaks down or, equivalently, the lifespan of a system;
- the Pareto distribution is often used to describe the distribution of incomes;
- the Cauchy distribution describes the points of impact of particles emitted in a beam;
- the beta distribution is used to fit distributions with known support.



It is also worth noting that the website [http://en.wikipedia.org/wiki/
List_of_probability_distributions](http://en.wikipedia.org/wiki/List_of_probability_distributions) describes many other distributions.

Warning



We only modelled phenomena involving several independent copies of the same random variable. In Chap. 14, we will present a statistical tool to model some phenomena involving several interacting random variables.

Memorandum

d--: mass or density function (e.g., `dnorm()`)
 p--: cumulative distribution function (e.g., `pchisq()`)
 q--: quantile function (e.g., `qt()`, `qf()`)
 r--: pseudorandom number generation (e.g., `runif()`)



Exercises

- 12.1-** Which R function would you use to generate numbers from a $\mathcal{N}(0, 1)$ distribution?
- 12.2-** Which R function would you use to generate numbers from a $\mathcal{N}(2, 10)$ distribution?
- 12.3-** Which R function would you use to calculate the quantiles of a χ^2 distribution?
- 12.4-** Which R function would you use to calculate the density of a Fisher distribution?
- 12.5-** Which R function would you use to calculate the quantiles of a Student distribution?
- 12.6-** How would you compute the probability that X lays between 3 and 5 given that $X \sim \mathcal{N}(4, 2)$?
- 12.7-** How would you calculate the quantile of order $p = 0.95$ of a $\mathcal{N}(0, 1)$?



Worksheet

Simulations

A- Study of the Distribution $f(x) = \frac{3}{2} \sqrt{x}$ on $[0, 1]$

- 12.1-** Check that $f(x)$ is a density, using the function `integrate()`.
- 12.2-** Simulate a sample of size 1,000 from the distribution defined by the density $f(x) = \frac{3}{2} \sqrt{x}$ over $[0, 1]$.
- 12.3-** Calculate the empirical mean and variance.
- 12.4-** Compare with the theoretical values.
- 12.5-** Calculate and compare the theoretical and empirical probabilities of the following classes:
 $[0, 0.30], [0.30, 0.50], [0.50, 0.70], [0.70, 0.85], [0.85, 1]$.

B- Study of the Generalized Pareto Distribution

Let X be a random variable following a generalized Pareto distribution $GP(\mu, \sigma, \xi)$. The density of this distribution is

$$f_X(x) = \frac{1}{\sigma} \left(1 + \frac{\xi(x - \mu)}{\sigma} \right)^{-\frac{1}{\xi} - 1}$$

with

$$x \geq \mu \text{ if } \xi \geq 0 \quad \text{and} \quad x \leq \mu - \sigma/\xi \text{ if } \xi < 0.$$

It is given that

$$\mathbb{E}(X) = \mu + \frac{\sigma}{1 - \xi} \quad (\xi < 1)$$

and

$$\text{Var}(X) = \frac{\sigma^2}{(1 - \xi)^2(1 - 2\xi)} \quad (\xi < 1/2).$$

We can simulate from X with the following formula:

$$X = \mu + \frac{\sigma(U^{-\xi} - 1)}{\xi}$$

where U is a uniform random variable over $[0, 1]$.

- 12.1-** Propose an R code to generate a sample of size n from a $GP(\mu, \sigma, \xi)$ distribution. Your source code should use the following variables: `n`, `mu`, `sigma` and `xi`.
- 12.2-** Simulate a sample of size $n = 1,000$ from the distribution $GP(0, 1, 1/4)$.
- 12.3-** Calculate the empirical mean and variance.
- 12.4-** Compare with the theoretical values.
- 12.5-** Repeat questions 2 to 3 with $n = 10,000$.
- 12.6-** Plot in red the density histogram of your sample. Take 500 equidistant classes and limit the display of the histogram to the interval $[0, 10]$ on the x-axis.
- 12.7-** Overlay the density plot of the $GP(0, 1, 1/4)$ distribution (in blue). Note that the plot is close to the histogram.

C- Uniform Distribution on a Square

- 12.1-** Simulate 1,000 observations from (X_1, X_2) which follow the uniform distribution over the square $[0, 1] \times [0, 1]$.
- 12.2-** Get an approximation of the probability that the distance between (X_1, X_2) and the nearest edge is less than 0.25.
- 12.3-** The same question for the distance to the nearest vertex.

- 12.4-** Try to identify the theoretical distribution of the variable distance to the nearest edge: expected value, variance and density.

D- Towards Modelling

A statistician believes that the world around us and the phenomena that occur are a large entanglement of random events, which can be **modelled** in a simplified way by random variables.

- 12.1-** Start with the simple, and classical example of a coin toss. The outcome of this experiment is the observation of HEAD or TAIL at each toss. This can be modelled by a random variable X with distribution a Bernoulli with parameter 1/2. This experiment can be reproduced with a computer. Create a function X which simulates a coin toss. You can toss your virtual coins a few times.
- 12.2-** We can also propose a **modelling** of the throw of a die. The outcome of this experiment is the observation of the number on the upper side of the die at each throw. If the die is not weighted, this can be modelled by a random variable X with distribution a discrete uniform over {1, 2, 3, 4, 5, 6}. This experiment can be reproduced with a computer. Create a function `throw.die()` using the function `sample()`. You can throw your virtual die a few times.
- 12.3-** To simulate the game of Yahtzee, we shall create a function `yahtzee()` which throws five virtual dice. Create this function using the parameters `size` and `replace` of the function `sample()`.
- 12.4-** Estimate the probability of getting a *yahtzee*, i.e. five identical dice in one throw (hint: use the functions `apply()`, `replicate()` and `unique()`). You should get a value close to $\frac{1}{6^4}$.

E- Box–Muller Theorem

Let U_1 and U_2 be two independent random variables uniform over the interval $[0, 1]$. The variables

$$Z_1 = \sqrt{-2 \log(U_1)} \cos(2\pi U_2)$$

$$Z_2 = \sqrt{-2 \log(U_1)} \sin(2\pi U_2)$$

are then two independent standard normal random variables.

- 12.1-** Generate $n = 1,000$ pairs of observations $(z_1, z_2)_1, \dots, (z_1, z_2)_n$ using this algorithm.
- 12.2-** Use the function `kde2d()` from package MASS to estimate the bivariate density of these data.
- 12.3-** Use the functions `spheres3d()` and `surface3d()` from the package `rgl` to plot these observations and the surface of the estimated bivariate density of these data. Also plot the surface of the density of a bivariate standard normal distribution. Note that you get a bell plot, typical of the bivariate normal.

Chapter 13

Confidence Intervals and Hypothesis Testing

Goals of this chapter

This chapter is a catalogue of R functions commonly used to get confidence intervals for usual parameters: mean, proportion, variance, median and correlation. We also present a catalogue of R functions to perform standard hypothesis testing. Furthermore, a few practical worksheets will help the reader understand how to interpret confidence intervals, as well as the various errors related to hypothesis testing.

— SECTION 13.1 —

Notations

Table 13.1 gives the notation we need to define the confidence intervals and hypothesis tests we introduce later in the chapter.

Table 13.1: Some notation for standard parameter estimation

Parameter	Notation	Estimator	Estimate	R function
Mean	μ	\bar{X}	\bar{x}	mean()
Variance	σ^2	$\hat{\sigma}^2$	$\hat{\sigma}^2$	var()
Median	m_e	\widehat{m}_e	\widehat{m}_e	median()
Correlation	ρ	$\hat{\rho}$	$\hat{\rho}$	cor()
Proportion	p	\hat{p}	\hat{p}	mean()

Table 13.2 gives the notation of quantiles we shall use.

Table 13.2: Notation of various quantiles of order p

Distribution	Notation	R function
Normal: $\mathcal{N}(0, 1)$	u_p	<code>qnorm(p)</code>
Student with n d.f.: $\mathcal{T}(n)$	t_p^n	<code>qt(p, df=n)</code>
Chi-squared with n d.f.: $\chi^2(n)$	q_p^n	<code>qchisq(p, df=n)</code>
Fisher with n and m d.f.: $\mathcal{F}(n, m)$	$f_p^{n,m}$	<code>qf(p, df1=n, df2=m)</code>

d.f.: degrees of freedom

— SECTION 13.2 —

Confidence Intervals

We are given a sample $\mathbf{x}_n = (x_1, \dots, x_n)^\top$ of random variables which depend on an unknown parameter θ which we wish to estimate. A random confidence interval with level (of confidence) $1-\alpha$ for θ consists in two random variables $A := a(\mathbf{x}_n; \alpha)$ and $B := b(\mathbf{x}_n; \alpha)$ such that

$$P[A \leq \theta \leq B] = 1 - \alpha.$$

The random variables A and B are the boundaries of this random confidence interval, usually noted

$$\text{CI}_{1-\alpha}(\theta) = [A, B].$$

When the sample is observed and we are given the data (x_1, \dots, x_n) , we note

$$ci_{1-\alpha}(\theta) = [a, b]$$

the resulting realization of the confidence interval, where $a = a(x_1, \dots, x_n; \alpha)$ and $b = b(x_1, \dots, x_n; \alpha)$. In the remainder of this chapter, by abuse of language, we shall not distinguish between a random confidence interval and its realization.

Finally, note that the correct way of interpreting a (random or realized) confidence interval will be given in the practical section. For now, we only give a catalogue of classical confidence intervals for the usual parameters: mean, proportion, variance, median and correlation.

13.2.1 Confidence Intervals for the Mean

- **Case of a large sample ($n > 30$) or of a small sample under the assumption of normality**

► *Definition:* A confidence interval at level $(1 - \alpha)$ for the mean μ is

$$ci_{1-\alpha}(\mu) = \left[\bar{x} - t_{1-\alpha/2}^{n-1} \frac{\hat{\sigma}}{\sqrt{n}}, \bar{x} + t_{1-\alpha/2}^{n-1} \frac{\hat{\sigma}}{\sqrt{n}} \right].$$

- **R instruction:** The confidence interval is given by the function `t.test()`.

- **Example of use:** Using the nutrition study, we wish to build a confidence interval of the mean of the weight of elderly people living in Bordeaux.

```
> t.test(weight,conf.level=0.9)$conf.int
[1] 65.16024 67.80436
attr(,"conf.level")
[1] 0.9
```

We get the confidence interval [65.16, 67.80] with confidence level 0.9.

- **Case of small samples**

- **Definition:** When no assumption is made about the data, we suggest using a bootstrap approach. Several bootstrap confidence intervals are given in [14].

- **R instruction:** The functions `boot()` and `boot.ci()` from package `boot()` can be used.

- **Example of use:** We are given a sample of cholesterol levels (in g/l) of ten women. This sample is representative of women living in France:

```
> chol.levels <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
Without the assumption of normality, we propose a confidence interval at the 95 % level of the mean cholesterol level of women living in France.
```

```
> require("boot")
> mymean <- function(x,indices) mean(x[indices])
> level.boot <- boot(chol.levels, mymean, R = 999,
+   stype = "i", sim = "ordinary")
> boot.ci(level.boot, conf = 0.95,type = c("norm","basic",
+   "perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
CALL:
boot.ci(boot.out = chol.levels.boot, conf = 0.95, type =
c("norm", "basic", "perc", "bca"))
Intervals:

| Level | Normal           | Basic            |
|-------|------------------|------------------|
| 95%   | ( 1.787, 2.366 ) | ( 1.770, 2.340 ) |



| Level | Percentile     | BCa            |
|-------|----------------|----------------|
| 95%   | ( 1.82, 2.39 ) | ( 1.83, 2.41 ) |

Calculations and Intervals on Original Scale
```

13.2.2 Confidence Intervals for a Proportion p

- **Case of large samples ($np \geq 5$ and $n(1 - p) \geq 5$)**

- **Definition:** A confidence level at level $(1 - \alpha)$ for the unknown proportion p is

$$ci_{1-\alpha}(p) = \left[\hat{p} - u_{1-\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}, \hat{p} + u_{1-\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \right].$$

- **Conditions for validity:** The required conditions ($np \geq 5$ and $n(1-p) \geq 5$) can be “checked” *a posteriori* by replacing p with the boundaries of the confidence interval. If the conditions are not verified, use the exact method given for the case of small samples.
- **R instruction:** The function `binom.approx()` from package `epitools` can be used.
- **Example of use:** We wish to build a confidence interval for the proportion of men (coded 2 below) amongst elderly people in Bordeaux, from the nutrition study.

```
> require("epitools")
> table(gender) # Sampling distribution of the gender
# variable.
gender
  1   2
 85 141
> binom.approx(141,226) [c("lower","upper")] # Calculating
# the CI
# with n=226.
      lower      upper
1 0.5607393 0.6870483
```

Warning



The function `prop.test()` also gives a confidence interval for the proportion, based on the score statistic.

- **Case of small samples: exact calculation**

- **Definition:** A confidence interval at level $(1 - \alpha)$ for proportion p comes from

$$n\hat{p} \sim \text{Bin}(n, p).$$

- **R instruction:** You can use the function `binom.test()`.

- **Example of use:** We return to the previous example to compute exactly the confidence interval for the proportion of men amongst elderly people in Bordeaux.

```
> binom.test(141,226)$conf # Computing the CI with
# n=226.
```

```
[1] 0.5572321 0.6872590
attr("conf.level")
[1] 0.95
```

Tip

The function `binom.exact()` from the package `epitools` returns the same confidence interval.



13.2.3 Confidence Intervals for a Variance

- **Case of samples under the assumption of normality**

- ▶ *Definition:* A confidence interval at level $(1 - \alpha)$ for the variance σ^2 is

$$ci_{1-\alpha}(\sigma^2) = \left[\frac{(n-1)\hat{\sigma}^2}{q_{1-\alpha/2}^{n-1}}, \frac{(n-1)\hat{\sigma}^2}{q_{\alpha/2}^{n-1}} \right].$$

- ▶ *R instruction:* The relevant function is `sigma2.test()`. This function is included in the package associated with this book.

- ▶ *Example of use:* We wish to build a confidence interval for the variance of the weight of elderly people living in Bordeaux, using the nutrition study.

```
> sigma2.test(weight, conf.level=0.9)$conf
[1] 124.8330 170.3277
attr("conf.level")
[1] 0.9
```

- **Case of samples without the assumption of normality**

When no assumption is made about the data, we suggest using a bootstrap method, as for the mean.

- ▶ *R instruction:* You can use the functions `boot()` and `boot.ci()` available in package `boot()`.

- ▶ *Example of use:* We return to the data about women's cholesterol levels and compute a confidence interval for the cholesterol level without assuming normality of the data.

```
> chol.levels <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
> require("boot")           # Load package boot.
> variance <- function(x,indices) var(x[indices])
```

```

> level.boot <- boot(chol.levels,variance,R=999,
+                      stype="i",sim="ordinary")
> boot.ci(level.boot,conf=0.95,type=c("norm",
+                                     "basic","perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
CALL:
boot.ci(boot.out=level.boot,conf=0.95,type=
+          c("norm","basic","perc","bca"))
Intervals:

| Level | Normal             | Basic              |
|-------|--------------------|--------------------|
| 95%   | ( 0.1060, 0.4412 ) | ( 0.1026, 0.4448 ) |



| Level | Percentile         | BCa                |
|-------|--------------------|--------------------|
| 95%   | ( 0.0521, 0.3943 ) | ( 0.1201, 0.4670 ) |

Calculations and Intervals on Original Scale
Some BCa intervals may be unstable

```

Note

Note that for large samples where you cannot assume normality, you can use an asymptotic approach. This is available in the package `asympTest`. Here is an example computing a confidence interval for the weight of elderly people living in Bordeaux using the nutrition study.



```

> require("asympTest")
> asymp.test(weight,par="var")$conf
[1] 121.6842 167.9196
attr(,"conf.level")
[1] 0.95

```

13.2.4 Confidence Intervals for a Median

- ▶ **Definition:** A confidence interval at level $(1 - \alpha)$ for the median m_e is

$$ci_{1-\alpha}(m_e) = [x_{(m_1)}, x_{(m_2+1)}],$$

where the $x_{(i)}$ are the ordered values of the sample of size n , m_1 is the smallest value such that $P(L \leq m_1) \geq \alpha/2$ and m_2 is the largest value such that $P(L \geq m_2) > \alpha/2$ with $L \sim \text{Bin}(n, 0.5)$.

- ▶ **R instruction:** A confidence interval for the median can be computed with the function `qbinom()`.
- ▶ **Example of use:** We return to the example of cholesterol levels and wish to build a 95 % confidence interval for the median cholesterol level of women living in France.

```
> chol.levels <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
> m1 <- qbinom(0.025,length(chol.levels),0.5)
> m2 <- qbinom(1-0.025,length(chol.levels),0.5)
> median.ci <- c(sort(chol.levels)[m1],sort(chol.levels[m2+1]))
> median.ci
[1] 1.6 2.0
```

Note

It is always possible to build a bootstrap confidence interval. For this example, the percentile method returns:

```
> chol.levels <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
> require("boot")
> mymedian <- function(x,indices) median(x[indices])
> levels.boot <- boot(chol.levels,mymedian,R=999,
+                         stype="i",sim="ordinary")
> levels.int <- boot.ci(levels.boot,conf=0.95,type="perc")
> levels.int$perc[4:5]
[1] 1.7 2.6
```



Also note that a non-parametric confidence interval is implemented in the function `wilcox.test()`.

```
> wilcox.test(chol.levels,conf.int=TRUE)$conf
[1] 1.70 2.45
attr("conf.level")
[1] 0.95
```

13.2.5 Confidence Intervals for a Correlation Coefficient

- *Definition:* A confidence interval at level $(1 - \alpha)$ for the correlation coefficient ρ is

$$ci_{1-\alpha}(\rho) = \left[\frac{\exp(2\hat{\theta}_{min}) - 1}{\exp(2\hat{\theta}_{min}) + 1}, \frac{\exp(2\hat{\theta}_{max}) - 1}{\exp(2\hat{\theta}_{max}) + 1} \right],$$

where $\hat{\theta}_{min} = \frac{1}{2} \ln \left(\frac{1+\hat{\rho}}{1-\hat{\rho}} \right) - u_{1-\alpha/2} \frac{1}{\sqrt{n-3}}$ and $\hat{\theta}_{max} = \frac{1}{2} \ln \left(\frac{1+\hat{\rho}}{1-\hat{\rho}} \right) + u_{1-\alpha/2} \frac{1}{\sqrt{n-3}}$.

- *R instruction:* You can use the function `cor.test()`.
- *Example of use:* We now wish to build a confidence interval for the correlation coefficient between the weight and height of elderly people living in Bordeaux, based on the nutrition study.

```
> cor.test(weight,height)$conf
[1] 0.5450122 0.7032775
```

```
attr(,"conf.level")
[1] 0.95
```

Warning

This confidence interval is only valid under the assumption of joint normality of the pair of variables, here (weight,height). If this assumption is not verified, you can always use bootstrap.

13.2.6 Summary Table for Confidence Intervals

Table 13.3 below provides a summary for confidence intervals.

Table 13.3: Summary for confidence intervals

Type	Condition for validity	R function
Proportion	$np \geq 5$ and $n(1 - p) \geq 5$ None	<code>prop.test(x)\$conf</code> <code>binom.test(x)\$conf</code>
Mean	$n > 30$ or normality	<code>t.test(x)\$conf</code>
Variance	Normality	<code>sigma2.test(x)\$conf</code>
Median	None	<code>wilcox.test(x)\$conf</code>
Correlation	Joint normal	<code>cor.test(x)\$conf</code>

SECTION 13.3

Standard Hypothesis Testing

We present briefly the philosophy of **hypothesis testing**.

Hypothesis tests propose a tool to aid decision or to validate an **assertion of interest** \mathcal{H}_1 (often called the “alternative hypothesis”). This statistical decision will be made based on a **decision rule**, which is intuitively built from a **test statistic** T (which depends on an observed sample). This decision may be wrong, so it is important to measure the risk of accepting \mathcal{H}_1 for each of the situations which are described by not- \mathcal{H}_1 . These risks of wrongly accepting \mathcal{H}_1 are called **risks of the first kind** and are defined under not- \mathcal{H}_1 . We then aim at controlling the worse (largest) of these risks, which is defined for a specific situation of not- \mathcal{H}_1 , usually noted \mathcal{H}_0 and called the “null hypothesis”. Thus any decision rule is associated with a maximum risk. Amongst potential decision rules, we choose the one which guarantees a low maximum risk α , specified in advance (usually 5%). This is called the **level of significance** α of the test.

When we apply this decision rule to an observed data set, it is interesting to know what is the smallest significance level that would have led to accepting \mathcal{H}_1 . In both

cases, the answer is what is called the *p*-value, defined as the (**maximal**) **risk level that would lead to accepting** \mathcal{H}_1 . The *p*-value is given by any statistical software, and the user can then compare this risk to a significance level α (fixed by the user or by some custom). Its interpretation is extremely simple: the smaller the *p*-value, the more reliable the decision (to accept the assertion of interest \mathcal{H}_1).

In the same context, it is worth mentioning the concept of **power of a test**: this function measures the probability of accepting \mathcal{H}_1 under any situation. These concepts will be explained and detailed, using the power of R, in the practical section. The remainder of this section gives a catalogue of classical statistical testing procedures and of R instructions to put them to use. Unless specified otherwise, the significance level of all tests presented hereafter is fixed at $\alpha = 5\%$.

Advanced users

Although we prefer our approach of hypothesis testing, which insists on the assertion of interest \mathcal{H}_1 , we present a mathematical curiosity which explains why some authors (often mathematicians) use the notation \mathcal{H}_0 to designate not- \mathcal{H}_1 . This makes sense from a mathematical point of view, since the logic of hypothesis testing is similar to proof by contradiction. Consider the simplistic example where we wish to prove $\mathcal{H}_1 : \mu_X > \mu_0$ based on a sample $\mathbf{X}_n = (x_1, \dots, x_n)$ of i.i.d. r.v.s following the distribution $\mathcal{N}(\mu_X, 1)$.

- (1) We reason by contradiction and suppose our hypothesis \mathcal{H}_1 is wrong. We are then in the contrary of \mathcal{H}_1 , which is noted **here** $\mathcal{H}_0 : \mu_X \leq \mu_0$. For example, this corresponds to the situation where $\mu_X = \tilde{\mu}$ for some value $\tilde{\mu}$ lesser than or equal to μ_0 .
- (2) We then wish to measure the plausibility of the contradiction hypothesis \mathcal{H}_0 . To this end, we shall measure how negative is the unknown spread $d = \mu_X - \mu_0$, estimated by the test statistic $T = \hat{\mu}_X - \mu_0 = \tilde{T} + (\tilde{\mu} - \mu_0)$, where $\tilde{T} = (\hat{\mu}_X - \tilde{\mu}) \sim \mathcal{N}(0, 1/n)$.
- (3) The data, which carry information about the value of μ_X since they were generated from a distribution with this expectation, give a realization t_{obs} of the statistic T. If \mathcal{H}_0 were true, then the random variable $T = \hat{\mu}_X - \mu_0$, which measures the spread between μ_X and μ_0 , would be unlikely to produce large values. Thus,

$$\begin{aligned}
 p(\tilde{\mu}) := P_{\mu_X=\tilde{\mu}}[T \geq t_{obs}] &= P_{\mu_X=\tilde{\mu}}[\tilde{T} + (\tilde{\mu} - \mu_0) \geq t_{obs}] \\
 &= P_{\mu_X=\tilde{\mu}}[\tilde{T} \geq t_{obs} - (\tilde{\mu} - \mu_0)] \\
 &\stackrel{\text{from (1)}}{\leq} P_{\mu_X=\tilde{\mu}}[\tilde{T} \geq t_{obs}] := \tilde{p} \\
 &\quad (\text{independent from } \tilde{\mu} \text{ here}).
 \end{aligned}$$



Note that $p(\tilde{\mu})$ (of which we do not know the numerical value) can be seen as an extension of the p -value for the situation $\tilde{\mu}$ and that \tilde{p} is only a computable upper bound, used to control this extended p -value. If the value $\tilde{p} = P[\mathcal{N}(0, 1/n) > t_{obs}]$ is very small (and thus the probability $p(\tilde{\mu})$ is even smaller), we get a (quasi) contradiction since the event $\{T \geq t_{obs}\}$ with probability $p(\tilde{\mu})$ did occur, even though we expected it to almost never happen under \mathcal{H}_0 . By contradiction, our hypothesis \mathcal{H}_1 is certainly true and \tilde{p} (smallest upper bound of the $p(\tilde{\mu})$, $\tilde{\mu} \leq \mu_0$, attained when $\tilde{\mu} = \mu_0$) expresses the strength of the contradiction (or of conviction) of not- \mathcal{H}_1 , or the risk of being wrong when deciding \mathcal{H}_1 . The probability \tilde{p} is nothing else than the p -value we defined earlier.

13.3.1 Parametric Tests

13.3.1.1 Tests of the Mean

- Comparing the theoretical mean and a reference value (case with one sample)

- ▶ *Description of the test:* Let X be a quantitative variable with theoretical mean μ and variance σ^2 . Using a sample of size n , we wish to compare the theoretical mean μ to a reference value μ_0 . The hypotheses of the test are $\mathcal{H}_0 : \mu = \mu_0$ and $\mathcal{H}_1 : \mu \begin{cases} > \\ \neq \\ < \end{cases} \mu_0$. Under \mathcal{H}_0 , the test statistic is

$$T = \sqrt{n} \left(\frac{\bar{X} - \mu_0}{\hat{\sigma}} \right) \sim T(n-1).$$

- ▶ *Conditions for validity:* Normally distributed data or large sample size ($n > 30$).

- ▶ *R instruction:* Use the function `t.test()`.

- ▶ *Example of use:* In the data set INTIMA-MEDIA, we wish to know whether people with a body mass index (BMI) above 30 have thicker intima-media on average than the population from which the sample is taken. We make the assumption that the theoretical mean of intima-media thickness in this population is 0.58 mm:

```
> BMI <- weight/((height/100)^2)
> measure1 <- measure[BMI>30]
```

```
> t.test(measure1, mu=0.58, alternative="greater")
One Sample t-test
data: measure1
t = 1.5272, df = 8, p-value = 0.08262
alternative hypothesis: true mean is greater than 0.58
95 percent confidence interval:
0.5715358      Inf
sample estimates:
mean of x
0.6188889
```

We cannot give a positive answer to the question, at the specified risk level $\alpha = 5\%$.

- **Comparing two theoretical means (case with two samples)**

► *Description of the test:* Let X_1 and X_2 be two quantitative variables (measuring the same quantity, but in two different populations). We assume that X_1 has theoretical mean μ_1 and variance σ_1^2 and that X_2 has theoretical mean μ_2 and variance σ_2^2 . Using estimations computed from two samples of respective size n_1 and n_2 from the two populations, we wish to compare μ_1 and μ_2 . The hypotheses of the test are $\mathcal{H}_0 : \mu_1 = \mu_2$ and $\mathcal{H}_1 : \mu_1 \begin{cases} > \\ \neq \\ < \end{cases} \mu_2$. Under \mathcal{H}_0 , the test statistic is

$$T = \frac{\bar{X}_1 - \bar{X}_2}{\hat{\sigma} \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \sim \mathcal{T}(n_1 + n_2 - 2),$$

where $\hat{\sigma}^2 = \frac{(n_1-1)\hat{\sigma}_1^2 + (n_2-1)\hat{\sigma}_2^2}{n_1+n_2-2}$, $\hat{\sigma}_1$ and $\hat{\sigma}_2$ being estimators of the variance in the two populations.

► *Conditions for validity:* Normally distributed variables X_1 and X_2 and equal variance.

► *R instruction:* Use the function `t.test()`.

► *Example of use:* We wish to see whether there is a significant difference of intima–media thickness between women with a physical activity and women with no such activity.

```
> measure.SPORT.1 <- measure[SORT==1&GENDER==2]
> measure.SPORT.0 <- measure[SORT==0&GENDER==2]
> t.test(measure.SPORT.1, measure.SPORT.0, var.equal=FALSE)
Welch Two Sample t-test
data: measure.SPORT.1 and measure.SPORT.0
t = -1.4693, df = 53.179, p-value = 0.1476
alternative hypothesis: true difference in means is not
equal to 0
```

```
95 percent confidence interval:
-0.07488130 0.01155649
sample estimates:
mean of x mean of y
0.5130435 0.5447059
```

We cannot give a positive answer to the question, at the specified risk level $\alpha = 5\%$.

Warning

However, we need to check the hypothesis of equal variances (see the next section). Thus the value of the argument `var.equal` in the function `t.test()` should be set depending on the result of the test of equal variances. The test statistic is then



$$T^* = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_2^2}{n_2}}}.$$

This statistic follows a Student distribution; the number of degrees of freedom can be computed using Satterthwaite's approximation. For large samples, the statistic T^* follows a $\mathcal{N}(0, 1)$ distribution.

- **Case of paired samples**

► *Description of the test:* We wish to compare the theoretical means of two random variables X_1 and X_2 based on two paired samples. To this end, we use the difference random variable $D = X_1 - X_2$, and we compare the theoretical mean $\delta = \mu_1 - \mu_2$ of D with the reference value 0. We are thus back to the case of a test of the mean for one sample. The hypotheses of the test are

$$\mathcal{H}_0 : \mu_1 - \mu_2 = 0 \text{ and } \mathcal{H}_1 : \mu_1 - \mu_2 \begin{cases} > \\ \neq \\ < \end{cases} 0. \text{ Under } \mathcal{H}_0, \text{ the test statistic is}$$

$$T = \sqrt{n} \frac{\bar{D}}{\hat{\sigma}} \sim \mathcal{T}(n - 1).$$

► *Conditions for validity:* Normally distributed data or large sample size ($n > 30$).

► *R instruction:* Use the function `t.test()` with the argument `paired=TRUE`.

► *Example of use:* We wish to compare the results from two labs for a specific examination. Both labs made the necessary measurement on 15 patients.

```
> dose.lab1 <- c(22,18,28,26,13,8,21,26,27,29,25,24,
+ 22,28,15)
```

```
> dose.lab2 <- c(25,21,31,27,11,10,25,26,29,28,26,23,
+                      22,25,17)
> t.test(dose.lab1,dose.lab2,paired=TRUE)
   Paired t-test
data: dose.lab1 and dose.lab2
t = -1.7618, df = 14, p-value = 0.0999
alternative hypothesis: true difference in means is not
equal to 0
95 percent confidence interval:
-2.0695338 0.2028671
sample estimates:
mean of the differences
-0.9333333
```

At the specified risk level $\alpha = 5\%$, we cannot decide that the two labs give different results on average.

Note

This test is valid when n is large or when the data can be assumed to be normally distributed. Otherwise, we advise you to use a non-parametric test such as the sign test or Wilcoxon's test (see Sect. 13.3.3).



13.3.1.2 Tests of Variance

- **Comparing the theoretical variance with a reference value (case with one sample)**

► *Description of the test:* Let σ^2 be the variance of a quantitative variable X.

The hypotheses of the test are $\mathcal{H}_0 : \sigma^2 = \sigma_0^2$ and $\mathcal{H}_1 : \sigma^2 \begin{cases} > \\ \neq \\ < \end{cases} \sigma_0^2$. Under \mathcal{H}_0 , the test statistic is

$$T = \frac{(n-1)\hat{\sigma}^2}{\sigma_0^2} \sim \chi^2(n-1).$$

► *Conditions for validity:* Variable X must follow a normal distribution.

► *R instruction:* You can use the function `sigma2.test()` from the package associated with this book.

► *Example of use:* A factory makes cans of weight μ with precision $\sigma^2 = 10$. We wish to show that the production line is faulty (the precision is not $\sigma^2 = 10$). Here are the weights from a series of 20 cans.

```
> weights <- c(165.1,171.5,168.1,165.6,166.8,170.0,168.8,
+           171.1,168.8,173.6,163.5,169.9,165.4,174.4,171.8,
+           166.0,174.6,174.5,166.4,173.8)
```

```
> sigma2.test(weights,var0=10)
   One-sample Chi-squared test for given variance
data: weights
X-squared = 24.2045, df = 19, p-value = 0.3768
alternative hypothesis: true variance is not equal to 10
95 percent confidence interval:
 7.367682 27.176225
sample estimates:
var of x
12.73924
```

We cannot give a positive answer to the question at the specified risk level $\alpha = 5\%$.

Note



If the data are not normally distributed, and for large samples, you can use the function `asymp.test(x, parameter="var", reference=10)`. This function is included in the package `asympTest`.

- **Comparing two theoretical variances (case with two samples)**

- ▶ *Description of the test:* This test is often useful as a prerequisite for other tests, such as the comparison of two means in the case with small samples. Indeed, in this case, the statistic is not the same depending on whether the variances of X_1 (variable for the first sample) and X_2 (variable for the second sample) can be considered as equal or not. The hypotheses of the test are $\mathcal{H}_0 : \sigma_1^2 = \sigma_2^2$ and $\mathcal{H}_1 : \sigma_1^2 \neq \sigma_2^2$. Under \mathcal{H}_0 , the test statistic is

$$T = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2} \sim F(n_1 - 1, n_2 - 1).$$

- ▶ *Conditions for validity:* X_1 and X_2 must be normally distributed.

- ▶ *R instruction:* You can use the function `var.test()`.

- ▶ *Example of use:* In the data set INTIMA-MEDIA, we wish to know whether within the population of women, there is a significant difference of variance of intima-media thickness between women who have a physical activity and those who do not.

```
> measure.SPORT.1 <- measure[SORT==1&GENDER==2]
> measure.SPORT.0 <- measure[SORT==0&GENDER==2]
> var.test(measure.SPORT.1,measure.SPORT.0)
   F test to compare two variances
data: measure.SPORT.1 and measure.SPORT.0
F = 0.303, num df = 22, denom df = 33, p-value =

```

```

0.00468
alternative hypothesis: true ratio of variances is not
equal to 1
95 percent confidence interval:
0.1431908 0.6815918
sample estimates:
ratio of variances
0.3029910

```

We can conclude that there is a significant difference in the variance of intima–media thickness between women who have a physical activity and those who do not, at the $\alpha = 5\%$ risk level.

Tip

Remember to use the function `asmp.test()` for large samples without the assumption of normality.



See also

For the comparison of more than two variances, see Bartlett's test, which will be presented in Sect. 15.1, page 509.



13.3.1.3 Tests of Proportion

- **Comparing a theoretical proportion to a reference value (case with one sample)**

► *Description of the test:* Let p be the unknown frequency of a trait in a given population. We observe data of presence/absence of this trait on individuals in a sample of size n in this population. The hypotheses of the test are $\mathcal{H}_0 : p = p_0$ and $\mathcal{H}_1 : p \begin{cases} > \\ \neq \\ < \end{cases} p_0$. Under \mathcal{H}_0 , the test statistic is

$$U = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} \sim \mathcal{N}(0, 1).$$

► *Conditions for validity:* The sample must be large enough: check that $np_0 \geq 5$ and $n(1 - p_0) \geq 5$.

► *R instruction:* You can use the function `prop.test()`.

► *Example of use:* Assume (and this really is no more than an assumption) that a case study called “Study of pregnant women in Abidjan” led to data

collection after a large campaign of information and prevention of HIV infection, which aimed at reducing the proportion of people infected by HIV, especially amongst people aged 18 to 25. Assume that the first aim is that the prevalence rate for pregnant women aged 18 to 25 be reduced to less than 10 %. Using the subsample of pregnant women aged 18 to 25, we therefore wish to know whether the HIV prevalence rate is less than $p_0 = 0.1$. The data set is available at the URL <http://www.biostatisticien.eu/springeR/aidsafrica.xls>.

```
> table(HIV[age<=25])
  0   1
137 10
> prop.test(10,147,0.1,alternative="less",correc=FALSE)
  1-sample proportions test without continuity
  correction
data: 10 out of 147, null probability 0.1
X-squared = 1.6697, df = 1, p-value = 0.09815
alternative hypothesis: true p is less than 0.1
95 percent confidence interval:
 0.0000000 0.1105720
sample estimates:
  p
0.06802721
```

We cannot give a positive answer to the question at the specified $\alpha = 5\%$ risk level.

- *Case of small samples:* In that case, the function `binom.test()` can be used for an exact calculation based on the binomial distribution.

```
> binom.test(10,147,0.1,alternative="less")
  Exact binomial test
data: 10 and 147
number of successes = 10, number of trials = 147,
p-value = 0.1208
alternative hypothesis: true probability of success is
less than 0.1
95 percent confidence interval:
 0.0000000 0.1126571
sample estimates:
probability of success
 0.06802721
```

Once again, we cannot give a positive answer to the question, at the specified $\alpha = 5\%$ risk.

• Comparing two theoretical proportions (case with two samples)

- *Description of the test:* Let p_1 (respectively p_2) be the unknown proportion of individuals with a given trait within a population \mathcal{P}_1 (respectively \mathcal{P}_2). We wish to compare p_1 and p_2 . To this end, we use the frequencies \hat{p}_1 and \hat{p}_2 of

this trait in two representative samples of the two populations, of respective sizes n_1 and n_2 . The hypotheses of the test are $\mathcal{H}_0 : p_1 = p_2$ and $\mathcal{H}_1 : p_1 \neq p_2$. Under \mathcal{H}_0 , the test statistic is

$$U = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\frac{\hat{p}(1-\hat{p})}{n_1} + \frac{\hat{p}(1-\hat{p})}{n_2}}} \sim \mathcal{N}(0, 1),$$

where $\hat{p} = \frac{n_1\hat{p}_1 + n_2\hat{p}_2}{n_1 + n_2}$.

- ▶ **Conditions for validity:** Large samples: check that $n_1\hat{p} \geq 5$, $n_1(1 - \hat{p}) \geq 5$, $n_2\hat{p} \geq 5$ and $n_2(1 - \hat{p}) \geq 5$.
- ▶ **R instruction:** Use the function `prop.test()`.

- ▶ **Example of use:** In the therapeutic test “Ditrame”, the underlying question is whether the treatment has an effect on the HIV status of the child. If not, then the HIV status of the child is independent of the treatment followed by the mother. To answer this question, we use the table of crossed variables Mother’s treatment group (TTTGRP) and Child’s HIV status (HIVSTATUS). We give the observed contingency table of these two variables. The data set is available at the URL http://www.biostatisticien.eu/springeR/TME_Africa.xls.

```
> table(TTTGRP, HIVSTATUS)
      HIVSTATUS
TTTGRP   0    1    9
      0 139  59   3
      1 152  41   7
> mytable <- as.matrix(table(TTTGRP, HIVSTATUS) [,c(2,1)])
> prop.test(mytable, correc=FALSE)
  2-sample test for equality of proportions without
  continuity correction
data: mytable
X-squared = 3.7574, df = 1, p-value = 0.05257
alternative hypothesis: two.sided
95 percent confidence interval:
-0.0004122543  0.1715013839
sample estimates:
prop 1    prop 2 
0.2979798 0.2124352
```

With a risk of being wrong less than 5 %, we can assert that the theoretical proportion p_1 of children with HIV in the non-treated group is greater than the theoretical proportion p_2 of children with HIV in the treated group.

13.3.1.4 Tests of Correlation

- Comparing a theoretical correlation coefficient with a reference value (case with one sample)

- *Description of the test:* Let ρ be the correlation coefficient of two quantitative variables X and Y. We wish to test the hypotheses $\mathcal{H}_0 : \rho = \rho_0$ and $\mathcal{H}_1 : \rho \neq \rho_0$. Under \mathcal{H}_0 , the test statistic is

$$U = \frac{Z - \mu_Z}{\sigma_Z} \sim \mathcal{N}(0, 1),$$

where $Z = \frac{1}{2} \ln \left(\frac{1 + \hat{\rho}}{1 - \hat{\rho}} \right)$, $\mu_Z = \frac{1}{2} \ln \left(\frac{1 + \rho_0}{1 - \rho_0} \right)$, $\sigma_Z^2 = \frac{1}{n - 3}$.

If we are interested in the linear association between X and Y (tested by taking $\rho_0 = 0$), then the test statistic under \mathcal{H}_0 is

$$T = \frac{\hat{\rho} \sqrt{n - 2}}{\sqrt{1 - R^2}} \sim \mathcal{T}(n - 2).$$

- *Conditions for validity:* The pair (X, Y) follows a joint normal distribution.
- *R instruction:* You can use the function `cor.test()` if you are testing the linear association between X and Y. For a value other than $\rho_0 = 0$, you can use the function `cor0.test()` available in the package associated with this book.
- *Example of use:* In the data set BMI-CHILD, we are interested in the linear association between height and weight.

```
> cor.test(weight, height)
Pearson's product-moment correlation
data: weight and height
t = 13.4327, df = 150, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.6570527 0.8036174
sample estimates:
cor
0.7389562
```

At the 5 % risk level, we can assert that there is a linear association between height and weight.

• Comparing two theoretical correlation coefficients (case with two samples)

- *Description of the test:* We are given two independent populations. Let ρ_1 be the correlation coefficient between X and Y in population 1 and ρ_2 the correlation coefficient between X and Y in population 2. We wish to test whether these two correlation coefficients are equal, based on two samples of sizes n_1 and n_2 . The hypotheses of the test are $\mathcal{H}_0 : \rho_1 = \rho_2$ and

$\mathcal{H}_1 : \rho_1 \begin{cases} > \\ \neq \\ < \end{cases} \rho_2$. Under \mathcal{H}_0 , the test statistic is

$$U = \frac{Z_1 - Z_2}{\sqrt{1/(n_1 - 3) + 1/(n_2 - 3)}} \sim \mathcal{N}(0, 1),$$

with $Z_1 = \frac{1}{2} \ln \left(\frac{1+\hat{\rho}_1}{1-\hat{\rho}_1} \right)$ and $Z_2 = \frac{1}{2} \ln \left(\frac{1+\hat{\rho}_2}{1-\hat{\rho}_2} \right)$, where $\hat{\rho}_1$ and $\hat{\rho}_2$ are the estimators of the correlation coefficients.

- ▶ *Conditions for validity:* In both populations, the pair (X, Y) must follow a joint normal distribution.
- ▶ **R instruction:** You can use the function `cor.test.2.sample()` available in the package associated with this book.
- ▶ *Example of use:* In the data set BMI-CHILD, we wish to compare the intensity of the relation height–weight between the group of girls and the group of boys.

```
> indg <- which(GENDER=="F") # To get the indices of girls.
> indb <- which(GENDER=="M") # To get the indices of boys.
> cor.test.2.sample(height[indg], weight[indg],
+                     height[indb], weight[indb])
$statistic
[1] -1.67379
$p.value
[1] 0.09417185
```

At the 5 % risk level, we cannot show a significant difference between the two coefficients of linear correlation.

13.3.2 Independence Tests

13.3.2.1 χ^2 Test for Independence

- ▶ *Description of the test:* Let X_1 and X_2 be two qualitative variables (or variables made qualitative by grouping): X_1 has l modalities and X_2 has c modalities. We wish to know whether these two variables are dependent, i.e. whether the modalities of variable X_2 are not distributed in the same way in each of the l subpopulations made of individuals taking each of the l modalities of variable X_1 .

We know the value of these modalities for n individuals; these data are usually given in a contingency table (or table of observed frequencies) and we compare this table to the theoretical contingency table for n individuals calculated under the assumption of independence of the two variables. The hypotheses are \mathcal{H}_0 : variables X_1 and X_2 are independent and \mathcal{H}_1 : the variables are not independent. Under \mathcal{H}_0 , the test statistic is

$$\chi^2 = \sum_{i=1}^l \sum_{j=1}^c \frac{(N_{ij} - t_{ij})^2}{t_{ij}^2} \sim \chi^2((c-1)(l-1)),$$

where N_{ij} is the observed count value and t_{ij} is the theoretical count value calculated under \mathcal{H}_0 , for the modality i of X_1 and j of X_2 .

- ▶ **Conditions for validity:** The theoretical values t_{ij} must be greater than 5; otherwise, you can use Yates' χ^2 (if the values are greater than 2.5 and only for 2×2 tables) or Fisher's exact test.
- ▶ **R instruction:** Use the function `chisq.test()` with the argument `correct =FALSE`.
- ▶ **Example of use:** We return to the example of the “Ditrame” study, where the underlying question is to know whether treatment has an effect on the HIV status of the child.

```
> mytable
  HIVSTATUS
  TTTGRP  1   0
          0 59 139
          1 41 152
> chisq.test(mytable, correct=FALSE)
  Pearson's Chi-squared test
data: mytable
X-squared = 3.7574, df = 1, p-value = 0.05257
```

At the 5 % risk level, we cannot show that treatment has an effect on the HIV status of the child.

Tip



You can also use the command `summary(table(HIVSTATUTS, TTTGRP))` to get the result of the χ^2 test of independence.

Note

You can perform a χ^2 test of mutual independence for $d \geq 2$ qualitative variables (see [5]). The χ^2 statistic is then

$$\chi^2 = \sum_{A \in \mathcal{I}_d} T_A \text{ with } \begin{cases} T_A = X_A^2 & \text{if } |A| = 2; \\ T_A = X_A^2 - \sum_{\{B \subset A; 1 < |B| < |A|\}} T_B & \text{if } |A| > 2. \end{cases}$$

In the last equation, \mathcal{I}_d is the family of all subsets of $\{1, \dots, d\}$ of size strictly greater than 1. Furthermore, if we are given the contingency array M of the d quantitative variables, then X_A^2 ($|A| \geq 2$) can be obtained with the instruction `summary(margin.table(M, A))$statistic` where A is the vector of indices of elements in A . Another advantage of this (orthogonal) decomposition of χ^2 , obtained by induction, is that each term T_A describes the mutual dependence of variables indexed by the set A .

The R function which performs this operation is `A.dep.tests()`. It is included in the package `IndependenceTests`.



13.3.2.2 Yates' χ^2 Test

- *Description of the test:* The χ^2 test with Yates' correction (or Yates' χ^2) should be used when you wish to perform a χ^2 test of independence for a 2×2 table, but at least one of the theoretical counts is less than 5; the theoretical counts should not be too small (> 2.5). The general setting is the same as the χ^2 test of independence. Under H_0 , the test statistic is

$$\chi^2 = \sum_{i=1}^l \sum_{j=1}^c \frac{(|N_{ij} - t_{ij}| - 0.5)^2}{t_{ij}^2} \sim \chi^2(1).$$

- *R instruction:* You can use the function `chisq.test()`.

- *Example of use:* In the data set INTIMA-MEDIA, we select people aged 50 or more and we wish to test whether their smoking habits are linked to gender.

```
> table.cont <- as.matrix(table(GENDER[AGE>=50],
+                               tobacco[AGE>=50]))
> chisq.test(table.cont)$expected # Table of theoretical
counts.
      0     1     2
1 2.88 0.48 2.64
2 9.12 1.52 8.36
> table.cont1 <- cbind(table.cont[,1],table.cont[,2]+
+                           table.cont[,3])
> chisq.test(table.cont1)
```

```
Pearson's Chi-squared test with Yates' continuity
correction
data: table.cont1
X-squared = 1.6732, df = 1, p-value = 0.1958
```

We cannot give a positive answer to the question at the 5 % risk level.

13.3.2.3 Fisher's Exact Test

- ▶ *Description of the test:* Fisher's exact test is used when the conditions for the χ^2 test of independence and Yates' χ^2 test are not verified, i.e. for small theoretical counts. Fisher's test can be used for tables with more than two rows or columns.
- ▶ *R instruction:* Use the function `fisher.test()`.
- ▶ *Example of use:* The definition of obesity is that an individual with BMI greater than 30 kg/m^2 is obese. We make the hypothesis that obesity is more common in people less than 50 years old. This means that the BMI variable is linked to age. We wish to answer this question using the “intima–media” study. To this end, we study the joint distribution of the following variables: under 50 / over 50 and $\text{BMI} < 30$ / $\text{BMI} > 30$. We get the following distribution:

```
> bmi <- weight/(height/100)^2
> obesity <- factor(bmi<30)
> levels(obesity) <- c("BMI>30", "BMI<30")
> age50 <- factor(AGE>=50)
> levels(age50) <- c("under 50", "over 50")
> table(age50,obesity)
      obesity
age50      BMI>30 BMI<30
  under 50      8    77
  over 50       1    24
> bmi.table <- as.matrix(table(obesity,age50))
> fisher.test(bmi.table,alternative="greater")
  Fisher's Exact Test for Count Data
data: bmi.table
p-value = 0.3478
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 0.3830018      Inf
sample estimates:
odds ratio
 2.477058
```

We cannot show that obesity is significantly more frequent amongst people under the age of 50.

Note

Fisher's exact test makes it possible to have a unilateral alternative hypothesis.

**Tip**

The function `CrossTable()` from the package `gmodels` performs independence tests (chi-square, Fisher exact and McNemar). It also produces outputs similar to those provided by SAS or SPSS.



13.3.3 Non-parametric Tests

13.3.3.1 Goodness-of-Fit Tests

- **Shapiro–Wilk test**

- *Description of the test:* The Shapiro–Wilk test is built specifically to test whether a continuous variable X is normally distributed. It is one of the most powerful tests of normality. The hypotheses are $\mathcal{H}_0 : X$ follows a normal distribution and $\mathcal{H}_1 : X$ does not follow a normal distribution. The test statistic is

$$W = \frac{T^2}{\hat{\sigma}^2},$$

where $\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$ and $T^2 = \frac{1}{n-1} \left[\sum_{i=1}^{[n/2]} a_i (X_{(n-i+1)} - X_{(i)}) \right]^2$.

The a_i are the coefficients of the Shapiro–Wilk table available in most books of statistical tables.

- *R instruction:* You can use the function `shapiro.test()`.
- *Example of use:* We return to the INTIMA–MEDIA data set used in the case “mean comparison with one sample”. We wish to invalidate the assumption of normality of intima–media thickness for people with BMI above 30.

```
> measure1
[1] 0.62 0.52 0.55 0.59 0.59 0.65 0.63 0.79 0.63
> shapiro.test(measure1)
Shapiro-Wilk normality test
data: measure1
W = 0.8835, p-value = 0.1708
```

We cannot show that the data are not normal, at the 5 % risk level.

- **χ^2 test for fit to a distribution**

The χ^2 test for fit to a distribution is used to check whether a qualitative variable does not follow a given theoretical distribution. It can be used to check whether a quantitative variable does not follow a given theoretical distribution, but in that case, it must first be grouped into classes to make it qualitative.

► *Description of the test:* Let X be a qualitative random variable with k modalities. We assume that X follows the distribution where modality i has probability p_i . Given a sample of size n , we test whether X follows the distribution defined by the p_i . The hypotheses of the test are $\mathcal{H}_0 : X$ follows the distribution defined by the p_i and $\mathcal{H}_1 : X$ does not follow this distribution. The test statistic is

$$\sum_{i=1}^k \frac{(N_i - np_i)^2}{np_i} \sim \chi^2(k - 1),$$

where N_i is the observed count for modality i .

- *Conditions for validity:* All theoretical counts np_i are greater than or equal to 5.
- *R instruction:* You can use the function `chisq.test()`.
- *Example of use:* A study on hypertension was performed, and we wish to test whether the sample of patients is representative of the general population from an ethnic point of view. We know that in the general population of Mauritius, the ethnic distribution is Hindu 51 %, Muslim 17 %, Creole 27 %, Chinese 3 % and Other 2 %. The data are available in the file <http://www.biostatisticien.eu/springeR/HTAen.xls>.

```
> table(ETHNIC)
ETHNIC
  1   2   3   4
225  77  99   1
> ni <- cbind(t(as.vector(table(ETHNIC))),0)
> chisq.test(ni,p=c(0.51,0.17,0.27,0.03,0.02))
  Chi-squared test for given probabilities
data:  ni
X-squared = 22.0659, df = 4, p-value = 0.0001945
```

At the 5 % risk level, we can conclude that the sample of patients is significantly not representative of the general population from an ethnic point of view.

- **Kolmogorov–Smirnov test for one sample**

► *Description of the test:* The aim of this test is the same as the χ^2 test for fit to a distribution. We wish to compare an empirical distribution to a completely

specified theoretical distribution. Let F_0 be the specified cumulative distribution function and F the cumulative distribution function of X . We are interested in the point at which the difference between the two cumulative distribution functions is greatest in absolute value, and we compare this value to the critical value given by the Kolmogorov–Smirnov table for one sample. The hypotheses of the test are $\mathcal{H}_0 : F = F_0$ and $\mathcal{H}_1 : F \begin{cases} > \\ \neq \\ < \end{cases} F_0$. The test statistic is

$$D = \sup_x |\hat{F}_{X_n}(x) - F_0(x)|,$$

where $\hat{F}_{X_n}(\cdot)$ is the empirical c.d.f. of the sample X_n .

- R instruction: Use the function `ks.test()`.

- Example of use: We return to the can factory example and try to invalidate the assumption of normality, which is necessary for the variance test. The cans are made with average weight $\mu = 170$ g and precision $\sigma^2 = 10$. We test the normality of the series of 20 cans made by the factory.

```
> weights <- c(165.1, 171.5, 168.1, 165.6, 166.8, 170.0, 168.8,
+             171.1, 168.8, 173.6, 163.5, 169.9, 165.4, 174.4,
+             171.8, 166.0, 174.6, 174.5, 166.4, 173.8)
> ks.test(weights, "pnorm", 170, sqrt(10))
One-sample Kolmogorov-Smirnov test
data: weights
D = 0.1942, p-value = 0.4376
alternative hypothesis: two-sided
```

At the 5 % risk level, we cannot show that the data are not normally distributed.

• Kolmogorov–Smirnov test for two samples

- Description of the test: The aim of this test is to compare two distributions F_1 and F_2 . We are interested in the point at which the difference between the two empirical cumulative distribution functions ($\hat{F}_{X_{n,1}}$ and $\hat{F}_{X_{n,2}}$) is greatest and we compare this value to the critical value given by the Kolmogorov–Smirnov table for two samples. The hypotheses of the test are $\mathcal{H}_0 : F_1 = F_2$ and $\mathcal{H}_1 : F_1 \begin{cases} > \\ \neq \\ < \end{cases} F_2$. The test statistic is

$$D = \sup_x |\hat{F}_{X_{n,1}}(x) - \hat{F}_{X_{n,2}}(x)|.$$

- R instruction: You can use the function `ks.test()`.

- Example of use: Using the data set INTIMA-MEDIA, we wish to see whether men who have quit smoking are younger than men who smoke. In the sample, there are twelve smokers and nine former smokers. The sample size is small; we do not know the distribution of the age variable in this population and therefore

cannot use a Student test to compare the means. The test is unilateral, since the hypothesis is that smokers are younger than former smokers.

```
> table(tobacco,GENDER)
   GENDER
tobacco 1 2
  0 32 40
  1  9  9
  2 12  8
> ks.test(AGE [GENDER==1&tobacco==1] ,AGE [GENDER=
+           =1&tobacco==2], alternative="greater")
Two-sample Kolmogorov-Smirnov test
data: AGE [GENDER == 1 & tobacco == 1] and AGE [GENDER == 1 &
tobacco == 2]
D^+ = 0.6389, p-value = 0.01502
alternative hypothesis: the CDF of x lies above that of y
```

In the population under study, we have shown at the 5 % risk level that men who have quit smoking are younger than men who smoke.

Tip

The package **Power** contains many other goodness-of-fit tests.

13.3.3.2 Tests of Position

- **Sign test or test of the median for one sample**

► *Description of the test:* We wish to compare the theoretical median of a quantitative variable X (denoted by m_e) to a reference value m_0 , with no assumption about the data. The hypotheses of the test are $\mathcal{H}_0 : m_e = m_0$ or equivalently $P(X - m_0 > 0) = 0.5$ and $\mathcal{H}_1 : m_e \begin{cases} > \\ \neq \\ < \end{cases} m_0$ or equivalently $P(X - m_0 > 0) \begin{cases} > \\ \neq \\ < \end{cases} 0.5$. Under \mathcal{H}_0 , the test statistic is

$$K \sim \text{Bin}(n, 0.5),$$

where K is the number of values strictly greater than m_0 . This test is therefore equivalent to a test of proportion.

► *R instruction:* You can use the function `prop.test()` or `binom.test()`.

► *Example of use:* The median price of one-bedroom flats in the Grenoble region in 2008 was 130,000 euros. We are given a sample of size $n = 32$ of prices of one-bedroom flats (in thousands of euros), from the free monthly magazine *L'Offre Immobilière*, issue number 91 (January 2009). We wish to know whether prices are increasing.

```

> m0 <- 130
> prices <- c(230.00,148.00,126.00,134.62,155.00,157.70,
+           160.00,225.00,125.00,109.00,157.00,115.00,
+           125.00,225.00,118.00,179.00,176.00,125.00,
+           123.00,180.00,151.00,120.00,143.00,170.00,
+           190.00,233.00,148.72,189.00,121.00,149.00,
+           225.00,240.00)
> sum(prices-m0>0)
[1] 22
> median(prices)
[1] 153
> prop.test(22,32,0.5,"greater")
  1-sample proportions test with continuity
  correction
data: 22 out of 32, null probability 0.5
X-squared = 3.7812, df = 1, p-value = 0.02591
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
 0.5266965 1.0000000
sample estimates:
  p
0.6875

```

We can conclude, at the 5 % risk level, that prices are increasing.

- **Sign test or median test for two independent samples**

► *Description of the test:* The median test is a test which compares the medians m_{e_1} and m_{e_2} of two quantitative variables X_1 and X_2 using data from two independent samples of these variables. The hypotheses of the test are $\mathcal{H}_0 : m_{e_1} = m_{e_2}$ and $\mathcal{H}_1 : m_{e_1} \begin{cases} > \\ \neq \\ < \end{cases} m_{e_2}$. Under \mathcal{H}_0 , we compute the common median $\widehat{m_e}$ for the two samples joined together. We then build a 2×2 table of counts of values which are smaller or greater than the median in the samples. This table is used as a χ^2 contingency table. We can then perform a χ^2 test, or (depending on the sample size) a χ^2 test with Yates' correction or a Fisher's exact test.

► *R instruction:* You can use the function `chisq.test()` or `fisher.test()`.

► *Example of use:* We return to the data set INTIMA-MEDIA and wish to test whether women who have quit smoking are younger than women who smoke. In this example, there are nine smokers and eight former smokers. The sample size is small and we cannot assume that the age variable is normally distributed in this population. We therefore cannot compare the means; it becomes interesting to perform a median test.

```

> Me <- median(AGE [GENDER==2&tobacco>0])
> tab.obs <- table(tobacco [tobacco>0&GENDER==2&AGE!=Me] ,
+                     AGE [GENDER==2&tobacco>0&AGE !=Me] >Me)

```

```

> rownames(tab.obs) <- c("Former smoker", "Smoker")
> colnames(tab.obs) <- c("AGE<ME", "AGE>ME")
> tab.obs

      AGE<ME AGE>ME
Former smoker       6      2
Smoker             2      6
> fisher.test(tab.obs, alt="greater")
    Fisher's Exact Test for Count Data
data: tab.obs
p-value = 0.06597
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
0.878644        Inf
sample estimates:
odds ratio
7.613556

```

At the 5 % risk level, we cannot assert that women who have quit smoking are younger than women who smoke.

- **Sign test for two paired samples**

► *Description of the test:* We wish to compare two paired quantitative series. We use the series of differences. We delete all concordant pairs (pairs for which the two values are equal) and keep only the n discordant pairs (non-zero difference between the two values). The sign test only takes into account the sign of the difference of discordant pairs. Let N^+ be the number of positive pairs and N^- the number of negative pairs. The test statistic is

$$S = \min(N^+, N^-) \sim \text{Bin}(n, 0.5).$$

► *R instruction:* You can use the function `prop.test()` or `binom.test()` after calculating the realization of S . Note that when $n \geq 20$, the distribution of S can be approximated by $\mathcal{N}(\frac{n}{2}, \frac{n}{4})$.

► *Example of use:* We return to the example of results from two labs, which we previously analysed using a test of comparison of means under the assumption of normality. The sign test does not require any assumption.

```

> dose.lab1 <- c(22, 18, 28, 26, 13, 8, 21, 26, 27,
+                 29, 25, 24, 22, 28, 15)
> dose.lab2 <- c(25, 21, 31, 27, 11, 10, 25, 26, 29,
+                 28, 26, 23, 22, 25, 17)
> dif <- (dose.lab1-dose.lab2)
> nminus <- sum(dif<0)
> nplus <- sum(dif>0)
> binom.test(min(nplus,nminus),nplus+nminus)
    Exact binomial test
data: min(nplus, nminus) and nplus + nminus
number of successes = 4, number of trials = 13, p-value
= 0.2668

```

```

alternative hypothesis: true probability of success is
not equal to 0.5
95 percent confidence interval:
0.0909204 0.6142617
sample estimates:
probability of success
0.3076923

```

At the 5 % risk level, we cannot conclude that the two labs give different results.

- **Wilcoxon or Mann–Whitney test for two independent samples**

- ▶ *Description of the test:* The Wilcoxon rank test is a non-parametric test which tests whether two distributions F_1 and F_2 are equal. The hypotheses of the test are $\mathcal{H}_0 : F_1 = F_2$ and $\mathcal{H}_1 : F_1 \begin{cases} > \\ \neq \\ < \end{cases} F_2$. The idea of the test is as follows: we merge the two series and sort the values from smallest to largest; we assign rank 1 to the smallest value, rank 2 to the next value and so on. We then calculate the score of each series by summing the ranks of the values from that series. Using the appropriate table, we then decide whether the scores are compatible with the hypothesis \mathcal{H}_0 of equal distributions. By convention, the test statistic S is the score of the sample with the smallest size. The statistic $W = S - \frac{n_0(n_0+1)}{2}$ is also used sometimes.
- ▶ *Conditions for validity:* If $n_0 = \min(n_1, n_2) \leq 10$, then the statistic W does not follow a standard distribution, but the corresponding probabilities are given in the Mann–Whitney/Wilcoxon table (function `pwilcox()`). If $\min(n_1, n_2) \geq 10$, we can consider that $W \sim \mathcal{N}\left(\mu_W = \frac{n_1 n_2}{2}; \sigma_W^2 = n_1 n_2 \frac{n_1 + n_2 + 1}{12}\right)$. If there are ties in the ranks, we also use a normal approximation, but with $\sigma_W^2 = \frac{n_1 n_2}{12} \times \left[(n_1 + n_2 + 1) - \sum_{j=1}^g \frac{t_j^3 - t_j}{(n_1 + n_2)(n_1 + n_2 - 1)} \right]$ where g is the number of groups of tied values and t_j the number of tied values in group j .
- ▶ *R instruction:* You can use the function `wilcox.test()`. Use the argument `exact=FALSE` for an approximate calculation using the normal approximation.
- ▶ *Example of use:* Using the dataset INTIMA-MEDIA, we would like to know whether men who smoke are older than men who have quit smoking. In this sample, there are nine smokers and twelve former smokers.

```

> wilcox.test(AGE[GENDER==1&tobacco==2], AGE[GENDER==1&
+           tobacco==1], exact=FALSE, alternative="greater")

```

```
Wilcoxon rank sum test with continuity correction
data: AGE[GENDER == 1 & tobacco == 2] and AGE[GENDER == 1 &
tobacco == 1]
W = 88.5, p-value = 0.007756
alternative hypothesis: true location shift is greater
than 0
```

At the 5 % risk level, we can conclude that men who smoke are older than men who have quit smoking.

Warning



To get the statistic W, you need to put the smallest sample first in the function `wilcox.test()`.

- **Wilcoxon's test for two paired samples**

- ▶ **Description of the test:** We wish to compare two paired quantitative series. We use the series of paired differences. We discard all concordant pairs (pairs for which the two values are equal) and keep only the discordant pairs (non-zero difference). Let n be the number of discordant pairs. We sort the values according to their absolute value and rank them. In case of a tie, we use the mean of the corresponding ranks. Let S^+ be the score of the series of positive differences and S^- the score of the series of negative differences. The scores are calculated in the same way as for the Wilcoxon test. The hypotheses of the test are $\mathcal{H}_0 : F_1 = F_2$ and $\mathcal{H}_1 : F_1 \begin{cases} > \\ \neq \\ < \end{cases} F_2$. We can use either S^+ or S^- as the test statistic.
- ▶ **Conditions for validity:** If $n \leq 30$, then the test statistic does not follow a standard distribution, but the corresponding probabilities can be found in the Wilcoxon table for paired series. If $n \geq 30$, then S^+ and S^- follow a $\mathcal{N}\left(\frac{n(n+1)}{4}; \frac{n(n+1)(2n+1)}{24}\right)$ distribution.
- ▶ **R instruction:** By default, R uses the statistic $v = S^+$ in the function `wilcox.test()` with argument `paired=TRUE`. Use the argument `exact=FALSE` to use the normal approximation.
- ▶ **Example of use:** We return to the example of the results from two labs, which we analysed with the sign test. This test is more powerful than the sign test, since it also takes into account the absolute value of the differences.

```
> dose.lab1 <- c(22, 18, 28, 26, 13, 8, 21, 26, 27,
+               29, 25, 24, 22, 28, 15)
> dose.lab2 <- c(25, 21, 31, 27, 11, 10, 25, 26, 29,
+               28, 26, 23, 22, 25, 17)
> wilcox.test(dose.lab1,dose.lab2,paired=TRUE,exact=FALSE)
Wilcoxon signed rank test with continuity
correction
```

```

data: dose.lab1 and dose.lab2
V = 22, p-value = 0.1047
alternative hypothesis: true location shift is not
equal to 0

```

At the 5 % risk level, we cannot conclude that the two labs give different results.

13.3.4 Memorandum of Standard Tests

This table lists all the tests we have introduced (Table 13.4).

Table 13.4: Standard tests

Nature	Data	Conditions for validity	R function
Parametric tests			
Mean	1 sample	$n > 30$ or normality	t.test(x,...)
	2 samples	Normality and equal variances	t.test(x,y,...)
	2 samples	Normality	t.test(x,y,var.equal=F)
	2 paired samples	$n > 30$ or normality	t.test(x,y,paired=T)
Variance	1 sample	Normality	sigma2.test(x,...)
	2 samples	Normality	var.test(x,y,...)
	2 samples	Large sample size	asymp.test(x,y,...)
Correlation	1 sample	Normality, $H_0 : \rho = \rho_0$	cor.test(x,y..)
	2 samples	Normality	cor.test.2.sample(x,y,...)
Proportion	1 sample	$np \geq 5$ and $n(1-p) \geq 5$	prop.test(x,...)
	1 sample		binom.test(x,...)
	2 samples	Large sample size	prop.test(x,y,...)
Independence tests			
χ^2 for independence Yates' χ^2 Fisher's exact test	Contingency table	Theoretical counts ≥ 5	chisq.test(.,correct=F)
	2×2 table	Theoretical counts ≥ 2.5	chisq.test()
	Contingency table		fisher.test()
Tests of fit to a distribution			
Shapiro–Wilk χ^2 of fit to a distribution Kolmogorov–Smirnov	1 sample		shapiro.test(x,...)
	1 sample		chisq.test()
	1 sample 2 samples	Theoretical counts ≥ 5	ks.test(x,.) ks.test(x,y)
Tests of position			
Median Sign test Mann–Whitney Mann–Whitney Wilcoxon	1 sample		binom.test(x,)
	2 samples		fisher.test(x,y,)
	2 paired samples		binom.test(x,y,paired=T)
	2 samples	$\min(n_1, n_2) \geq 10$	wilcox.test(x,y,exact=F)
	2 samples	$\min(n_1, n_2) \leq 10$	wilcox.test(x,y)
	2 paired samples		wilcox.test(x,y,paired=T)

— SECTION 13.4 —

Other Tests

Many other tests are available in R. For example, you can get a first list of tests with the following instruction:

```
> apropos(".test")
[1] ".runRUnitTestsGdata"    ".valueClassTest"
[3] "ansari.test"           "as.krandtest"
[5] "as.randtest"           "as.rtest"
[7] "asymp.test"            "bartlett.test"
[9] "binom.test"             "Box.test"
[11] "chisq.test"             "cor.test"
[13] "cor.test.2.sample"     "cor0.test"
[15] "file_test"              "fisher.test"
[17] "fligner.test"           "friedman.test"
[19] "gpuTtest"                "kruskal.test"
[21] "ks.test"                  "mantel.randtest"
[23] "mantel.rtest"            "mantelhaen.test"
[25] "mauchley.test"           "mauchly.test"
[27] "mcnemar.test"             "mood.test"
[29] "multispaci.randtest"      "multispaci.rtest"
[31] "oneway.test"              "ormidp.test"
[33] "pairwise.prop.test"       "pairwise.t.test"
[35] "pairwise.wilcox.test"      "plot.krandtest"
[37] "plot.randtest"             "plot.rtest"
[39] "poisson.test"              "power.anova.test"
[41] "power.prop.test"           "power.t.test"
[43] "PP.test"                   "print.krandtest"
[45] "print.randtest"             "print.rtest"
[47] "procuste.randtest"         "procuste.rtest"
[49] "prop.test"                  "prop.trend.test"
[51] "quade.test"                 "randtest"
[53] "randtest.amova"             "randtest.between"
[55] "randtest.cca"                 "randtest.coinertia"
[57] "randtest.discrimin"          "randtest.pcaiv"
[59] "randtest.pcaivortho"        "randtest.rlq"
[61] "rate2by2.test"                 "rtest"
[63] "rtest.between"                 "rtest.discrimin"
[65] "rtest.niche"                  "RV.rtest"
[67] "RVdist.randtest"               "shapiro.test"
[69] "sigma2.test"                  "t.test"
[71] "tab2by2.test"                  "var.test"
[73] "var0.test"                     "wilcox.test"
```

Note

Other tests are available in packages. For example, package **nortest** includes various tests of fit to normality. The functions associated with these tests are



```
> require("nortest")
> ls("package:nortest")
[1] "ad.test"        "cvm.test"       "lillie.test"
[4] "pearson.test"   "sf.test"
```

Memorandum

`t.test()`: test and confidence interval for the mean
`var.test()`: test for equality of variances
`prop.test()`: approximate confidence interval for a proportion
`binom.test()`: exact confidence interval for a proportion
`cor.test()`: test and confidence interval for correlation
`chisq.test()`: χ^2 test
`fisher.test()`: Fisher's test of independence
`ks.test()`: Kolmogorov–Smirnov test of fit to a distribution
`shapiro.test()`: Shapiro–Wilk test of normality
`med.test()`: median test
`wilcox.test()`: Wilcoxon's test of position
`boot()`: package for bootstrap



Exercises

- 13.1- Which function would you use to get the quantiles of a binomial distribution?
- 13.2- What is the use of the function `pnorm()`?
- 13.3- Give the R instruction to get a confidence interval for the mean using a sample of size 50.
- 13.4- Explain the difference between the functions `prop.test()` and `binom.test()`.
- 13.5- Name two functions which compare two cumulative distribution functions using two samples.
- 13.6- Which function is used to test whether a sample follows a normal distribution?
- 13.7- Which functions are used to test the dependence of qualitative variables?
- 13.8- Which package includes functions to compute confidence intervals using bootstrap?
- 13.9- Which formal argument of the function `t.test()` is used for a paired test?
- 13.10- What is the difference between a χ^2 for independence and a χ^2 for fit to a distribution? How would you perform these two tests in R?



Worksheet

A- Study of Confidence Intervals

The aim of this practical is to understand how to interpret a confidence interval. Indeed, for a confidence interval at level $1 - \alpha$ of an unknown parameter, it is incorrect to say that there is a $100 \times (1 - \alpha)\%$ chance that the parameter lies in

the realized interval. The unknown parameter has a unique value which does not change: the probability that it lies in the realized interval is either 0 or 1. However, it is correct to say that there is a 5 % risk of being wrong by stating that the parameter lies in the realized confidence interval.

- *Study of the confidence interval for the mean:*

- 13.1-** Simulate $M = 50,000$ samples of size $n = 20$ from a normal distribution with mean $\mu = -1.2$ and variance $\sigma^2 = 2$.
- 13.2-** For each sample, compute a 90 % confidence interval of the mean μ .
- 13.3-** Calculate the proportion of intervals which contain the value $\mu = -1.2$. What do you observe?
- 13.4-** Repeat this procedure for the value $\mu = 1$, with samples of size $n = 100$ from a $\chi^2(1)$ distribution.
- 13.5-** Same question with $n = 10$ and a $\chi^2(1)$ distribution. What do you observe? How do you explain this?
- 13.6-** Simulate a sample of size $n = 20$ from a normal distribution with mean $\mu = -1.2$ and variance $\sigma^2 = 2$. Calculate a 95 % confidence interval for μ .
- 13.7-** Repeat this operation for samples of increasing size: $n = 50; 100; 1,000; 10,000; 100,000$. What do you observe?
- 13.8-** For each of the six samples above, calculate the observed value of the statistic for a Student test of hypothesis $\mathcal{H}_1 : \mu \neq 0$, as well as the p -value of the test (at significance level 5 %). What do you observe? How do you explain this?
- 13.9-** For a sample of size $n = 100,000$, calculate a 95 % confidence interval and compute the p -value of the test of hypothesis $\mathcal{H}_1 : \mu \neq -1.1$. Compare this p -value to the one you found in the previous question with $n = 50$. What do you conclude?

- *Study of confidence intervals from bootstrap:*

- 13.1-** Simulate $M = 500$ samples of size $n = 20$ from an exponential distribution with expectation $1/\lambda = 10$.
- 13.2-** For each sample, calculate a 90 % confidence interval of the mean $1/\lambda$ using the bootstrap method.
- 13.3-** Using the method described above, check the level of the confidence interval.
- 13.4-** Compare this level to the one you get with a standard confidence interval for the mean (procedure `t.test()`).

B- Study of Risks in Hypothesis Testing

The aim of this practical is to explore the risks associated with hypothesis testing:

- 1) $P[\text{accept } \mathcal{H}_1 | \mathcal{H}_0 \text{ is true}] = \alpha$, the risk of deciding \mathcal{H}_1 when in reality \mathcal{H}_0 is true;

- 2) $P[\text{reject } \mathcal{H}_1 | \mathcal{H}_1 \text{ is true}] = \beta$, the risk of not deciding \mathcal{H}_1 when in reality \mathcal{H}_1 is true.

- *Study of risk of the first kind:*

- 13.1-** Simulate $M = 500$ samples of size $n = 20$ following a normal distribution with mean $\mu = 4$ and variance $\sigma^2 = 1.2$.
- 13.2-** For each sample, perform a Student test for hypotheses $\mathcal{H}_0 : \mu = 4$ and $\mathcal{H}_1 : \mu \neq 4$ at level $\alpha = 5\%$.
- 13.3-** Count how many times you decide \mathcal{H}_1 . What result did you expect?
- 13.4-** Increase the number M of simulations.
- 13.5-** Repeat this procedure for the value $\mu = 1$, with samples of size $n = 100$ from a $\chi^2(1)$ distribution.
- 13.6-** Same question with $n = 10$. What do you observe? How do you explain this?
- 13.7-** In fact, for each of the $M = 500$ samples, we make the decision either to accept \mathcal{H}_1 or to reject it. Let d_j ($1 \leq j \leq M$) be the random variable which takes the value 1 if we decide to accept \mathcal{H}_1 and 0 otherwise (based on the j th sample). The variable d_j follows a Bernoulli distribution with parameter $p = P[d_j = 1] = P[\text{accept } \mathcal{H}_1]$. The variables d_j are independent; hence, the variable $d = \sum_{j=1}^M d_j$ follows a binomial distribution $\text{Bin}(M, p)$. It counts how many times we accept \mathcal{H}_1 . If \mathcal{H}_0 is true and the test is well constructed (well chosen critical value), then we should have $p = P[\text{accept } \mathcal{H}_1 | \mathcal{H}_0 \text{ is true}] = \alpha$. Calculate a 95 % confidence interval for parameter p and conclude.

- *Study of power:*

- 13.1-** Simulate $M = 500$ samples of size $n = 20$ from a normal distribution with mean $\mu = 5$ and variance $\sigma^2 = 1.2$.
- 13.2-** On each sample, perform a Student test between $\mathcal{H}_0 : \mu = 4$ and $\mathcal{H}_1 : \mu \neq 4$ at level $\alpha = 5\%$.
- 13.3-** Count the number of times you accept \mathcal{H}_1 . Estimate the power of the test for the situation in \mathcal{H}_1 where $\mu = 5$.
- 13.4-** Increase the size of the sample to $n = 100$ and estimate the power of the test. What do you observe?
- 13.5-** Repeat the procedure for the test $\mathcal{H}_0 : \mu = 1$ and $\mathcal{H}_1 : \mu \neq 1$, with samples of size $n = 100$ from a $\chi^2(2)$ distribution.
- 13.6-** Same question with $n = 10$. What do you observe? How do you explain this?

C- A Few Practical Examples

- *Cow study:* The quantity of bacteria per cm^3 of milk from eight different cows is estimated after milking and 24 h later. We wish to test whether the quantity of bacteria significantly increases with time.

Cow	Just after milking	24 h after milking
1	12,000	11,000
2	13,000	20,000
3	21,500	31,000
4	17,000	28,000
5	15,000	26,000
6	22,000	30,000
7	11,000	16,000
8	21,000	29,000

13.1- Answer the question under the assumption of normality of the data.

13.2- Answer the question using a sign test.

13.3- Answer the question using a Mann–Whitney test.

- *East German athletes:* In the 1970s, female athletes from East Germany were well known for their corpulence. The Olympic ethics committee of the time, intrigued by this “masculinity”, required the services of Dr. Volker Fischbach. He selected nine female athletes with identical morphological characteristics, then measured the quantity of androgens (hormones which stimulate male characteristics) per litre of blood. The results are 3.22 3.07 3.17 2.91 3.40 3.58 3.23 3.11 3.62.

13.1- Given that for women who do not use performance-enhancing drugs, the mean quantity of androgens is 3.1, propose a way to test whether East German athletes used such drugs (assume normality of data).

13.2- What was Dr. Fischbach conclusion?

- *Drinking and driving:* To study the effect of alcohol on reflexes, 14 subjects went through a test of dexterity before and after drinking 100 mL of wine. The before and after scores are given in the following table (these are reaction times: a higher score means slower reflexes).

Subject	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Before	57	54	62	64	71	65	70	75	68	70	77	74	80	83
After	55	60	68	69	70	73	74	74	75	76	76	78	81	90

Propose a way to test whether alcohol has an effect on reflexes (assume normality of the data).

- *Speed of light:* In 1879, American physicist Michelson performed several experiments to check the speed of light c proposed by French physicist Cornu in 1876. Cornu proposed a value of 299,990 km/s. Michelson got the 20 following measures (we subtracted 299,990 from Michelson’s values, to avoid having to handle large numbers):

850 740 900 1,070 930 850 950 980 980 880 1,000 980 930 1050 960 810 1,000
1,000 960 960.

These 20 observations can be considered as the observed values of 20 random variables with identical but unknown mean μ . If the conditions to measure the speed of light are satisfactory, then it is reasonable to assume that μ is the true speed of light.

- 13.1-** Plot the data. Comment.
- 13.2-** Test the normality of the data.
- 13.3-** Perform a Student test to check whether Michelson's measures invalidate the value of c proposed by Cornu.
- 13.4-** What value for the speed of light could Michelson propose after these 20 experiments?

• *Cholesterol levels:* 17 people with the same illness are randomly assigned to two groups. The first group receives a placebo A and the second group receives a treatment B containing a vitamin. The relevant measure is capillary resistance. We get the following measurements:

Group A: 46.3 ; 42.5 ; 43.0 ; 43.9 ; 42.0 ; 41.5 ; 41.6 ; 44.4 ; 40.7

Group B: 47.1 ; 44.5 ; 45.8 ; 49.0 ; 44.6 ; 43.7 ; 44.5 ; 47.4

Assuming that the measure of capillary resistance is normally distributed, what can you conclude?

• *Treatment-death independence:* Two small groups of animals are infected by a virulent germ. The first group receives chemotherapy; the second group is not treated. We measure mortality after eight days in both groups.

	With treatment	Without treatment	Total
Dead	0	9	9
Survived	8	3	11
Total	8	12	20

Is mortality independent of treatment?

• *Number of patients in the emergency ward:* To study the variation of the number of emergency cases in a hospital, the number of patients was counted for the months of June, July and August. The results are

	June	July	August
Number of patients	1,500	1,600	1,450
Number of emergency patients	675	720	610

Can we conclude that the proportion of emergency cases is the same every month?

Chapter 14

Simple and Multiple Linear Regression

Goals of this chapter

This chapter is a brief introduction to simple and multiple linear regression and how to use this method in a real context (see [41] for a more complete presentation). We present the relevant R commands and use a real data set as a connecting thread as we present the key concepts for this method. We treat the case of qualitative explanatory variables, as well as interaction of explanatory variables. We discuss model validation with a study of residuals and mention the issue of collinearity. We also present a few methods for variable selection.

— SECTION 14.1 —

Introduction

In most situations, we study the relationship between a variable of interest Y (often quantitative) and one or several variable(s) X_1, X_2, \dots, X_k , with the aim of explaining the variations of the variable of interest. Variable Y is called the **explained** variable (or dependent variable, or response variable). Variables X_1, X_2, \dots, X_k are called **explanatory** variables (or independent variables); in epidemiology, they represent risk or confounding factors. Thus, multivariate analysis models, and specifically linear regression models, allow us to:

- Take into account simultaneously several factors which could explain the variation or the distribution of variable Y
- Study the role of effect modification or confounding of one or several factor(s)
- Predict the value or distribution of the explained variable knowing the value of the explanatory variables

To introduce the key concepts of linear regression, we shall first introduce simple linear regression: we consider a quantitative explained variable Y and a single quantitative explanatory variable X , though in principle X could also be qualitative. In the second part, we shall introduce multiple linear regression, which is used to study the relation between a dependent quantitative variable Y and several (quantitative or qualitative) explanatory variables X_1, X_2, \dots, X_k .

Connecting Thread Example: “Weight at Birth” Study

We return to the data set `BIRTH-WEIGHT` which we presented in Chap. 2. We wish to explain the variability of child weight at birth as a function of characteristics of the mother, of family history and of behaviour during pregnancy. The explained variable is weight at birth (quantitative variable `BWT`, expressed in grammes); the explanatory variables are described in Chap. 2.

► Loading the Data

```
> myfile <- "http://www.biostatisticien.eu/springeR/Weight_birth.csv"
> mydata <- read.table(myfile, header=TRUE, sep="\t")
```

The weight of the mother is expressed in pounds. We first transform the `data.frame` to recode this variable in kilogrammes (1 pound = 0.45359237kg).

```
> mydata <- transform(mydata, LWT=LWT*0.4535923)
> attach(mydata) # Access variable names.
```

SECTION 14.2

Simple Linear Regression

14.2.1 Aim and Model

► Aim

We wish to “explain” the variations of a quantitative variable Y (e.g., child weight at birth, noted `BWT`) using an explanatory variable X , also quantitative (e.g., weight of the mother, noted `LWT`).

► Model

It is written as

$$Y = \beta_0 + \beta_1 X + \epsilon,$$

where ϵ represents the noise of the model, which is assumed to be normally distributed, with expectation zero and variance $\text{Var}(\epsilon|X) = \sigma^2$. The (unknown) parameters of the regression model are β_0 , β_1 and σ^2 .

Warning

The assumption of normality of the noise ϵ is necessary to get the distribution of the estimators and thus perform hypothesis testing on the parameters of the model. However, this assumption is not very important, since it is not necessary when the data size is large.



14.2.2 Fitting Data

► Graphical Inspection

To study the relationship between child weight at birth and weight of the mother, we first draw the scatterplot of the points (child weight; mother weight) using the instruction `plot(BWT~LWT)`.

```
> plot(BWT~LWT,xlab="Mother weight",ylab="Child weight at birth")
```

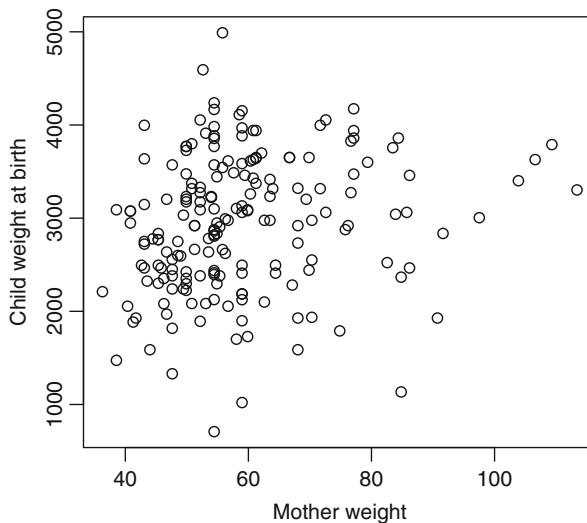


Fig. 14.1: Scatter plot of child weight (in grammes) against mother weight (in kilogrammes)

We observe a slight increase in child weight when mother weight increases, although this relationship is not very clear (Fig. 14.1).

► Parameter Estimation

We now study the following model:

$$BWT_i = \beta_0 + \beta_1 LWT_i + \epsilon_i, \quad i = 1, \dots, n,$$

where the ϵ_i are independent random variables with expectation zero and **constant variance** σ^2 , for all i .

Let us note bwt_i and lwt_i the observed values of the random variables BWT and LWT respectively. The estimates $\hat{\beta}_0$ of β_0 and $\hat{\beta}_1$ of β_1 are obtained by minimizing the least squares criterion:

$$S(\alpha_0, \alpha_1) = \sum_{i=1}^n (bwt_i - \alpha_0 - \alpha_1 lwt_i)^2.$$

The relevant R function for this operation is `lm()` (abbreviation of *linear models*). The main parameter of this function is a formula, symbolized by a tilde \sim , used to specify the relationship between BWT and LWT.

```
> modell <- lm(BWT ~ LWT, data=mydata) # We get an object
                                         # of class "lm".
> modell
Call:
lm(formula = BWT ~ LWT, data = mydata)
Coefficients:
(Intercept)          LWT
2369.672            9.765
```

The above R output gives the least squares estimates of β_0 and β_1 . For the above example, we get $\hat{\beta}_0 = 2,369.672$ and $\hat{\beta}_1 = 9.765$.

We can now draw the regression line on the scatter plot, using the function `abline()` (Fig. 14.2):

```
> plot(BWT~LWT,xlab="Mother weight",ylab="Child weight")
> abline(modell,col="blue")
```

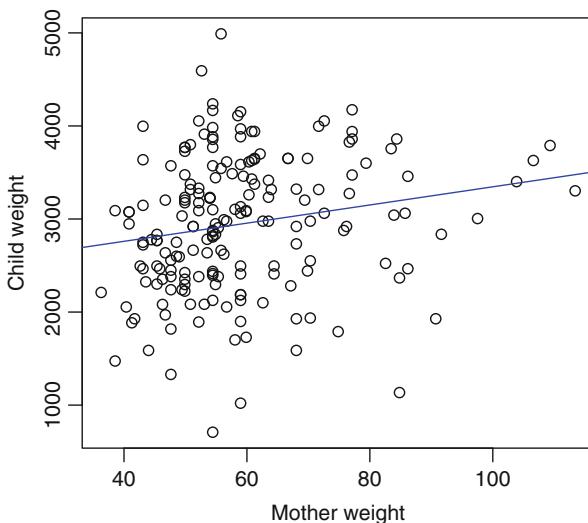


Fig. 14.2: Least squares regression line on the scatter plot of child weight (in grammes) against mother weight (in kilogrammes)

► Tests on Parameters

Note that the function `lm()` performs a complete analysis of the linear model and that you can get a summary of the calculations related to the data set with the function `summary()`.

```
> res <- summary(modell) # Results.
> res
Call:
lm(formula = BWT ~ LWT, data = mydata)
Residuals:
    Min      1Q  Median      3Q     Max 
-2192.184 -503.627   -3.910   508.250  2075.529 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2369.672   228.431  10.374 <2e-16 ***
LWT          9.765     3.777   2.586   0.0105 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
Residual standard error: 718.2 on 187 degrees of freedom
Multiple R-squared:  0.03452,    Adjusted R-squared:  0.02935 
F-statistic: 6.686 on 1 and 187 DF,  p-value: 0.01048
```

Here is a description of the information in this output.

- **Call:** formula used in the model.
- **Residuals:** descriptive analysis of residuals $\hat{e}_i = \hat{y}_i - y_i$. Later in this chapter, we shall see that residuals are used to validate the assumptions of the regression model.
- **Coefficients:** this table has four columns:
 - **Estimate** gives the estimates of the parameters of the regression line.
 - **Std. Error** gives the estimate of the standard deviation of the estimators of the regression line.
 - **t value** gives the realization of Student's test statistic associated with the hypotheses $\mathcal{H}_0 : \beta_i = 0$ and $\mathcal{H}_1 : \beta_i \neq 0$.
 - **Pr(>|t|)** gives the *p*-value of Student's test.
- **Signif. codes:** codes for significance levels.
- **Residual standard error:** an estimate of the standard deviation of the noise σ and the associated degree of freedom $n - 2$.
- **Multiple R-squared:** coefficient of determination r^2 (percentage of variation explained by the regression).
- **Adjusted R-squared:** adjusted r_a^2 (of limited interest for simple linear regression).
- **F-statistic:** realization of Fisher's test statistic associated with the hypotheses $\mathcal{H}_0 : \beta_1 = 0$ and $\mathcal{H}_1 : \beta_1 \neq 0$. The associated degrees of freedom (1 and $n - 2$) are given, as is the *p*-value.

Note

Note that the values of all these fields can be extracted. For example, to get the four columns of the field `Coefficients`, use

```
> res$coefficients
            Estimate Std. Error   t value    Pr(>|t|) 
(Intercept) 2369.672063 228.430647 10.373705 3.328580e-20
LWT          9.764856   3.776573   2.585639 1.048072e-02
```

The other suffixes are given by



```
> names(res)
[1] "call"           "terms"         "residuals"      
[4] "coefficients"  "aliased"        "sigma"        
[7] "df"             "r.squared"     "adj.r.squared"
[10] "fstatistic"    "cov.unscaled"
```

Also, note that the function `coefficients()` gives $\hat{\beta}_0$ and $\hat{\beta}_1$ directly from `model1`.

```
> coefficients(model1)
(Intercept)      LWT
2369.672063    9.764856
```

► Analysis of Variance Table

In simple linear regression, Fisher's test (the statistic of which is given in `F-statistic`) is equivalent to Student's test for the regression slope. The relationship $F\text{-statistic} = t^2$ holds and the p -values of the tests are equal. Fisher's test is often associated with an analysis of variance table, given by function `anova()`.

```
> anova(model1)
Analysis of Variance Table
Response: BWT
              Df  Sum Sq Mean Sq F value    Pr(>F)  
LWT          1 3448881 3448881  6.6855 0.01048 *  
Residuals 187 96468171 515873
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

► Interpreting the Results for the “Weight at Birth” Study

- The test associated with the intercept β_0 of the model is significant (p -value < 0.05); it is therefore advised to keep the intercept (β_0) in the model. However, the intercept in this regression has no meaning. It might be better to use a regression on the variable mother weight, centred beforehand. In that case, β_0 would represent the mean child weight for mothers who have weight equal to the mean weight of observed mothers.

Tip

The instruction to perform linear regression without an intercept is `lm(y~x-1)` or equivalently `lm(y~0+x)`.



- The linear relationship between BWT and LWT is proven by the result of Student's test on coefficient β_1 . The p -value < 0.05 indicates a significant linear relationship between child weight and mother weight.
- The percentage of variance explained by the regression (r^2) is 0.035. Only 3.5 % of the variability of child weight is explained by mother weight. We therefore need to add to the model other explanatory variables (multiple linear regression), to increase the model's predictive power.
- The estimate of the slope indicates that the difference between the average weights of babies whose mothers have weights different by 1 kg is 9.765 g.

Tip

To get an estimate by confidence interval of the regression coefficients, you can use the function `confint()`.

```
> confint(model1)
              2.5 %      97.5 %
(Intercept) 1919.039836 2820.30429
LWT          2.314692   17.21502
```



The 95 % confidence interval for β_1 is $ci_{95\%}(\beta_1) = [2.31, 17.22]$.

14.2.3 Confidence and Prediction Intervals for a New Value

► Definition

Consider a new observation x_0 of variable X for which we have not observed the corresponding value y_0 of the response variable Y. This value y_0 is unknown, since it is not observed, and is a realization of the random variable $Y_0 = \beta_0 + \beta_1 x_0 + \epsilon_0$. The **predictor** of Y_0 for the new value x_0 is given by

$$\hat{Y}_0^p = \hat{\beta}_0 + \hat{\beta}_1 x_0.$$

We can also propose a **prediction interval** at level $1 - \alpha$ for Y_0 , by finding two random bounds such that the random variable Y_0 falls in the interval with probability $1 - \alpha$:

$$PI_{1-\alpha}(Y_0|x_0) = \left[\hat{Y}_0^p \pm t_{1-\alpha/2}^{(n-2)} \hat{\sigma} \sqrt{1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}} \right].$$

Note that the realization $\hat{y}_0^P = \hat{\beta}_0 + \hat{\beta}_1 x_0$ is called the **prediction** of the unobserved value $y_0 = \beta_0 + \beta_1 x_0 + \epsilon_0$.

Similarly, note that an estimator of the fixed and unknown value $\mathbb{E}(Y_0|X = x_0) = \beta_0 + \beta_1 x_0$ is given by

$$\hat{\mathbb{E}}(Y_0|X = x_0) := \hat{Y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0.$$

We can also give a confidence interval at level $1 - \alpha$ for $\mathbb{E}(Y_0|X = x_0)$:

$$CI_{1-\alpha}(\beta_0 + \beta_1 x_0) = \left[\hat{Y}_0 \pm t_{1-\alpha/2}^{(n-2)} \hat{\sigma} \sqrt{\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}} \right].$$

Warning



Note that $\hat{Y}_0^P = \hat{Y}_0$, but do not confuse the prediction interval for Y_0 and the confidence interval for the mean value $\mathbb{E}(Y_0|X = x_0) = \beta_0 + \beta_1 x_0$.

► R Instruction

The function to define the prevision interval and confidence interval for a new value x_0 is `predict()`.

► Example with the Study “Weight at Birth”

We calculate the prediction of the weight of a baby whose mother weighs $lwt = 56$ kg.

```
> lwt0 <- 56
> predict(modell,data.frame(LWT=lwt0),interval="prediction")
   fit      lwr      upr
1 2916.504 1495.699 4337.309
```

For the confidence interval of the mean value of the weight of babies with a mother weighing 56 kg:

```
> predict(modell,data.frame(LWT=lwt0),interval="confidence")
   fit      lwr      upr
1 2916.504 2811.225 3021.783
```

We now represent the confidence interval and prediction interval for a series of new values of the mother's weight (Fig. 14.3):

```

> x <- seq(min(LWT),max(BWT),length=50)
> predint <- predict(modell,data.frame(LWT=x),interval=
+                               "prediction") [,c("lwr","upr")]
> confint <- predict(modell,data.frame(LWT=x),interval=
+                               "confidence") [,c("lwr","upr")]
> plot(BWT~LWT,xlab="Mother weight",ylab="Child weight")
> abline(modell)
> matlines(x,cbind(confint,predint),lty=c(2,2,3,3),
+            col=c("red","red","blue","blue"),lwd=c(2,2,1,1))
> legend("bottomright",lty=c(2,3),lwd=c(2,1),
+        c("confidence","prediction"),col=c("red","blue"))

```

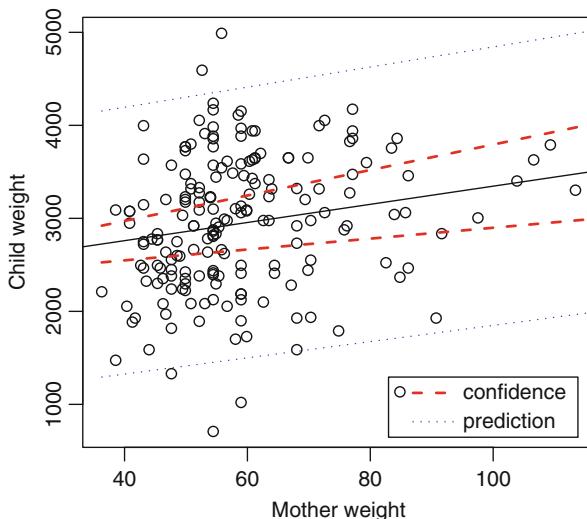


Fig. 14.3: Visualization of confidence and prediction intervals

14.2.4 Analysis of Residuals

► Checking Model Assumptions

Analysis of residuals consists in examining whether the underlying assumptions of the linear model are verified. Various plots of the residuals can be used to detect rather easily whether the assumptions on the errors ϵ_i are respected:

- Plotting the histogram of residuals to check for normality. The QQ-plot is another approach. You can also apply the Jarque–Bera test of normality on residuals (function `jarque.bera.test()` in package `tseries`).

- Plotting the residuals $\hat{\epsilon}_i$ as a function of the predicted values \hat{y}_i . When all the model assumptions are verified, residuals and predicted values are uncorrelated. This plot should have no particular structure. This plot also gives indications on the validity on the linearity assumption and on the homogeneity of error variance. The plot of $\hat{\epsilon}_i$ against \hat{y}_i should show a uniform spread of the residuals following a horizontal line on either side of the x axis.
- For the assumption of independence of the Y_i : if the values Y_i are measured on different, unrelated individuals, the assumption of independence is in principle verified. However, if the values of Y_i represent measures of a quantity at different points in time (e.g., every month), the assumption of independence may not be verified. This assumption can be checked by examining the autocorrelation of the residuals, either graphically by plotting the residuals $\hat{\epsilon}_i$ or using statistical tests such as the Durbin–Watson test (function `dwttest()` in package `lmtest`).

► Example of Analysis of Residuals in Study “Weight at Birth”

We briefly present an analysis of residuals for the model under study, even though this is not very relevant given the low predictive power of the model ($r^2 = 3.5\%$).

We first examine the assumption of normality (Fig. 14.4):

```
> par(mfrow=c(1,2))
> hist(residuals(model1), main="Histogram")
> qqnorm(resid(model1), datax=TRUE) # Caution: normalized
                                         # quantiles on the y axis.
```

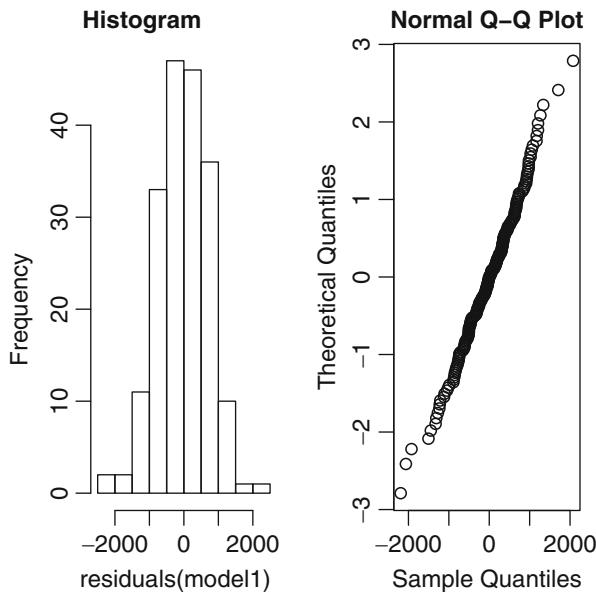


Fig. 14.4: Graphical inspection of normality of residuals

```
> require("tseries")
> jarque.bera.test(residuals(modell)) # Jarque-Bera test of
# normality.
    Jarque Bera Test
data: residuals(modell)
X-squared = 1.7877, df = 2, p-value = 0.4091
```

This test does not lead us to rejecting the assumption of normality of errors. The QQ-plot also suggests normal errors since the observed quantiles and theoretical quantiles (given by a normal distribution) form a straight line. We therefore decide to accept the assumption of normality.

We now examine the plot of residuals as a function of predicted values. The next figure shows a scatter plot of residuals correctly spread, symmetrical about the x axis: the conditions of the model seem valid (Fig. 14.5).

```
> plot(residuals(modell)~fitted(modell),
+       xlab="Predicted values", ylab="Residuals")
```

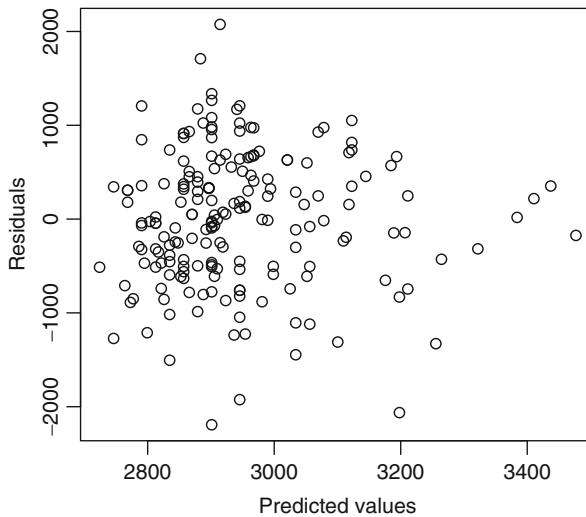
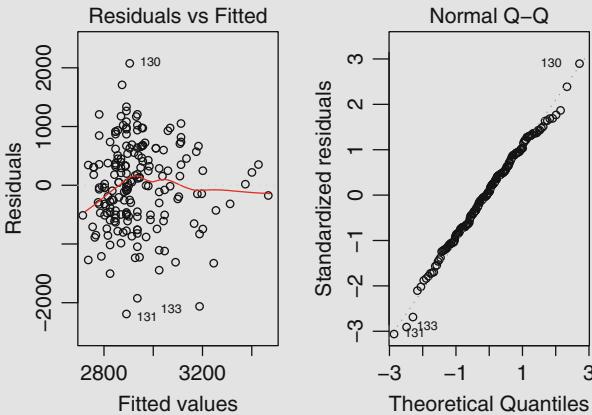


Fig. 14.5: Plot of residuals as a function of predicted values

Note

Some diagnostic plots can be obtained with the instructions:

```
> par(mfrow=c(1,2))
> plot(model1,1:2,col.smooth="red")
```



Note that the instruction `plot(model1)` can draw six plots; some of these are for detection of outliers. We shall return to this topic in Sect. 14.3.

14.2.5 Student's Tests for Means and Linear Model

It is interesting to note that the one-sample t-test and the classical or paired two-sample t-tests are special cases of simple linear regression models.

- **One-sample t-test**

Let Y_1, \dots, Y_n be i.i.d. $\mathcal{N}(\mu, \sigma^2)$ random variables. We want to test

$$\mathcal{H}_0 : \mu = \mu_0 \text{ versus } \mathcal{H}_1 : \mu \neq \mu_0,$$

for some given reference value μ_0 . This can be performed by testing $\beta = 0$ in the model

$$Y_i - \mu_0 = \beta + \epsilon_i,$$

where the ϵ_i are i.i.d. $\mathcal{N}(0, \sigma^2)$, $1 \leq i \leq n$. Let us see this on an example:

```
> n <- 100
> y <- rnorm(n, mean=0)
> mu0 <- 0.1
> t.test(y, mu=mu0) [[3]]
```

```
[1] 0.08523596
> summary(lm(y-mu0 ~ 1)) [[4]][1,4]
[1] 0.08523596
```

- **Two-sample t-test**

Let $X_1^{(1)}, \dots, X_{n_1}^{(1)}$ be i.i.d. $\mathcal{N}(\mu_1, \sigma^2)$ random variables and $X_1^{(2)}, \dots, X_{n_2}^{(2)}$ be i.i.d. $\mathcal{N}(\mu_2, \sigma^2)$ random variables. The two samples are supposed to be independent. We want to test

$$\mathcal{H}_0 : \mu_1 = \mu_2 \text{ versus } \mathcal{H}_1 : \mu_1 \neq \mu_2.$$

This can be done by testing $\beta_1 = 0$ in the model

$$Y_i = \mu_1 + \beta_1 A_i + \epsilon_i,$$

where the ϵ_i are i.i.d. $\mathcal{N}(0, \sigma^2)$ random variables and where we define

$$A_i = \begin{cases} 0 & \text{si } 1 \leq i \leq n_1, \\ 1 & \text{si } n_1 + 1 \leq i \leq n_1 + n_2, \end{cases}$$

and

$$Y_i = \begin{cases} X_i^{(1)} & \text{si } 1 \leq i \leq n_1, \\ X_{i-n_1}^{(2)} & \text{si } n_1 + 1 \leq i \leq n_1 + n_2. \end{cases}$$

Let us see this on an example:

```
> n1 <- 100 ; n2 <- 150
> x1 <- rnorm(n1,mean=0) ; x2 <- rnorm(n2,mean=0.1)
> y <- c(x1,x2) ; a <- c(rep(0,n1),rep(1,n2))
> t.test(x1,x2,var.equal=TRUE) [[3]]
[1] 0.3996454
> summary(lm(y ~ a)) [[4]][2,4]
[1] 0.3996454
```

- **Two-sample paired t-test**

Let $X_1^{(1)}, \dots, X_n^{(1)}$ be i.i.d. $\mathcal{N}(\mu_1, \sigma^2)$ random variables and $X_1^{(2)}, \dots, X_n^{(2)}$ be i.i.d. $\mathcal{N}(\mu_2, \sigma^2)$ random variables, measured on the same statistical units. We want to test

$$\mathcal{H}_0 : \mu_1 = \mu_2 \text{ versus } \mathcal{H}_1 : \mu_1 \neq \mu_2.$$

This can be done by testing $\beta = 0$ in the model

$$Y_i = \beta + \epsilon_i,$$

where the ϵ_i are i.i.d. $\mathcal{N}(0, \sigma^2)$ random variables and where $Y_i = X_i^{(1)} - X_i^{(2)}$, $1 \leq i \leq n$. Let us see this on an example:

```
> n <- 100
> x1 <- rnorm(n,mean=0) ; x2 <- rnorm(n,mean=0.1)
```

```
> t.test(x1,x2,var.equal=TRUE,paired=TRUE) [[3]]
[1] 0.4541976
> summary(lm(x1~x2 ~ 1)) [[4]][1,4]
[1] 0.4541976
```

14.2.6 Summary

The table below presents the main functions to use for simple linear regression between the response variable Y and the explanatory variable X (Table 14.1).

Table 14.1: Main R functions for simple linear regression

R instruction	Description
plot(Y~X)	Scatter plot
lm(Y~X)	Estimation of the linear model
summary(lm(Y~X))	Description of results of the model
abline(lm(Y~X))	Draw the estimated line
confint(lm(Y~X))	Confidence interval for regression parameters
predict()	Function for predictions
plot(lm(Y~X))	Graphical analysis on residuals

SECTION 14.3

Multiple Linear Regression

14.3.1 Aim and Model

► Aim

We wish to study the variations of a quantitative variable Y (the explained or response variable, assumed to be random) as a function of p ($p > 1$) explanatory variables X_1, X_2, \dots, X_p (also called independent variables). The explanatory variables can be all quantitative, all qualitative (in which case we fall back on ANOVA, presented in Chap. 15) or a mixture of quantitative and qualitative variables. In this last case, the multiple linear regression model is also called ANCOVA.

► Model

The multiple linear regression model is written as

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon,$$

where ϵ is the random noise term of the model, often assumed to be normally distributed with expectation zero and variance σ^2 and independent of the X_j . The (unknown) parameters of the regression model are $\beta_0, \beta_1, \dots, \beta_p$ and σ^2 .

14.3.2 Fitting Data

Consider a data set from the following model:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} + \epsilon_i \quad i = 1, \dots, n,$$

where X_{ij} is the j th explanatory variable for individual i and the errors ϵ_i are independent random variables such that $\mathbb{E}(\epsilon_i) = 0$ and $\text{Var}(\epsilon_i | \mathbf{X}_i) = \sigma^2$ with $\mathbf{X}_i = (X_{i1}, \dots, X_{ip})$. The observed counterpart of this model can be written in matrix form:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

$$\text{with } \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & & & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix} \text{ and } \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}.$$

The regression parameters $\boldsymbol{\beta}$ are estimated by minimizing (in $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \dots, \alpha_p)$) the ordinary least squares:

$$S(\boldsymbol{\alpha}) = \sum_{i=1}^n (y_i - \alpha_0 - \alpha_1 x_{i1} - \alpha_2 x_{i2} - \cdots - \alpha_p x_{ip})^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\alpha}\|^2.$$

This gives the least squares estimator $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$.

Warning

We assume that $\mathbf{X} : n \times (p + 1)$ is a full rank matrix ($\text{rank}(\mathbf{X}) = p + 1 < n$). This implies that $\mathbf{X}^\top \mathbf{X}$ is invertible. This condition can be evaluated using R with the command `qr(X)$rank`. If this condition is not satisfied, it is possible to use other methods, not shown in this book, as ridge regression, lasso regression, regression on principal components or PLS regression.



► Graphical Inspection

Connecting thread example: Regression of child weight at birth as a function of mother age, weight and smoking status during pregnancy.

In this study, we note X_1 the age of the mother (variable AGE), X_2 the weight of the mother (variable LWT), X_3 the smoking status of the mother (variable SMOKE) and Y the child weight at birth (response variable BWT); we can write the following regression equation:

$$\mathbb{E}(BWT|AGE, LWT, SMOKE) = \beta_0 + \beta_1 AGE + \beta_2 LWT + \beta_3 SMOKE.$$

Before estimating the model, we present a scatter plot of all pairs of variables (Fig. 14.6).

```
> add.cor <- function(x,y) {
+   usr <- par("usr"); on.exit(par(usr))
+   par(usr = c(0, 1, 0, 1))
+   text(0.5,0.5,round(cor(x,y),2))
+ }
```

```
> newdata <- cbind(BWT,LWT,AGE,SMOKE)
> pairs(newdata,lower.panel=panel.smooth,upper.panel=add.cor)
```

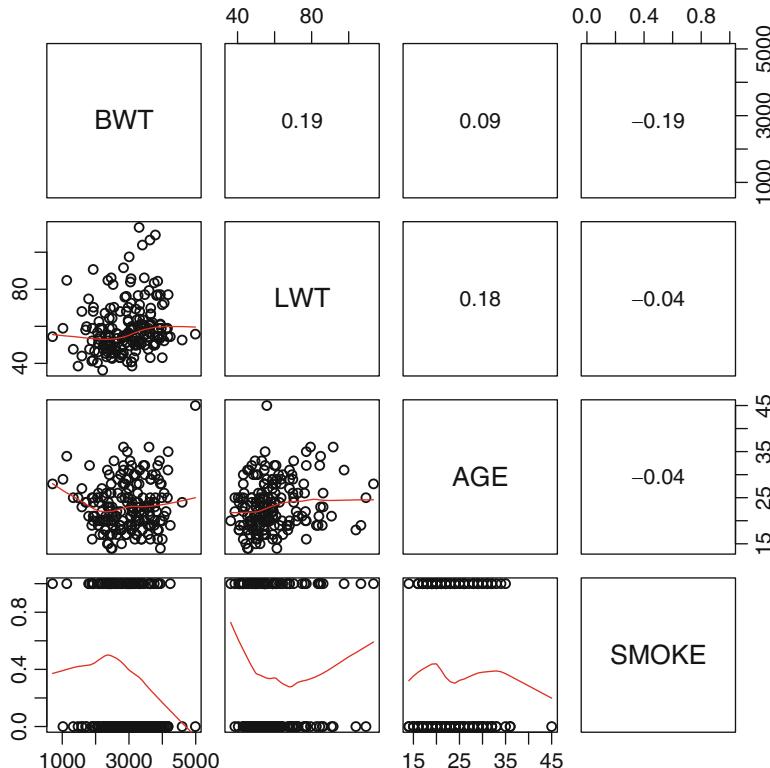


Fig. 14.6: Scatter plot of all pairs of variables

The purpose of this plot is twofold. It is used to visualize the relationship between the response variable and each of the explanatory variables, but also the correlation between explanatory variables. This second point is useful, amongst other things, because it helps notice collinearity issues (see the section on this topic, Sect. 14.3.7).

► Parameter Estimation

As for simple linear regression, the model is estimated using function `lm()`:

```
> model2 <- lm(BWT~AGE+LWT+as.factor(SMOKE))
> model2
Call:
lm(formula = BWT ~ AGE + LWT + as.factor(SMOKE))
Coefficients:
(Intercept)                AGE                  LWT
2362.720                  7.093                 8.860
as.factor(SMOKE)1          -267.213
```

► Tests on Parameters

Tests on parameters are performed by function `summary()`.

```
> summary(model2)
Call:
lm(formula = BWT ~ AGE + LWT + as.factor(SMOKE))
Residuals:
    Min      1Q  Median      3Q     Max 
-2069.89 -433.18   13.67  516.45 1813.75 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2362.720   300.687   7.858 3.11e-13 ***
AGE          7.093     9.925   0.715  0.4757    
LWT          8.860     3.791   2.337  0.0205 *  
as.factor(SMOKE)1 -267.213   105.802  -2.526  0.0124 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 708.8 on 185 degrees of freedom
Multiple R-squared:  0.06988,    Adjusted R-squared:  0.05479 
F-statistic: 4.633 on 3 and 185 DF,  p-value: 0.003781
```

The results output by `summary()` are presented in the same fashion as for simple linear regression. Parameter estimates are given in the column `Estimate`.

The realizations of Student's test statistics associated with the hypotheses $\mathcal{H}_0 : \beta_i = 0$ and $\mathcal{H}_1 : \beta_i \neq 0$ are given in column `t value`; the associated *p*-values are in column `Pr(>|t|)`. **Residual standard error** gives the estimate of σ and the number of associated degrees of freedom $n - p - 1$. The coefficient of determination r^2 (**multiple R-squared**) and an adjusted version (**adjusted R-squared**) are given, as are the realization of Fisher's global test statistic (**F-statistic**) and the associated *p*-value.

► Analysis of Variance Table

The analysis of variance table is given by function `anova()`:

```
> anova(model2)
Analysis of Variance Table
Response: BWT
            Df  Sum Sq Mean Sq F value Pr(>F)
AGE          1 806927 806927 1.6063 0.20661
LWT          1 2970564 2970564 5.9133 0.01598 *
as.factor(SMOKE) 1 3204339 3204339 6.3787 0.01239 *
Residuals   185 92935223 502353
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note

Fisher's global F test is used to test the global joint contribution of all explanatory variables in the model to "explaining" the variations of Y . The null hypothesis is $\mathcal{H}_0 : \beta_1 = \beta_2 = \dots = \beta_p = 0$ (under the linear model, the p explanatory variables give no useful information to predict Y). The assertion of interest is \mathcal{H}_1 : at least one of the coefficients β_j ($j = 1, 2, \dots, p$) is significantly different from zero (at least one of the explanatory variables is associated with Y after adjusting for the other explanatory variables).



► Interpreting Results from Study “Weight at Birth”

Given the result of Fisher's global test (p -value = 0.003781), we can conclude that at least one of the explanatory variables is associated with child weight at birth, after adjusting for the other variables. The individual Student tests indicate that:

- Mother weight is linearly associated with child weight, after adjusting for age and smoking status, with risk of error less than 5% (p -value = 0.0205). At same age and smoking status, an increase of 1 kg in mother weight corresponds to an increase of 8.860 g of average child weight at birth.
- The age of the mother is not significantly linearly associated with child weight at birth when mother weight and smoking status are already taken into account (p -value = 0.20661).
- Weight at birth is significantly lower for a child born to a mother who smokes, compared to children born to non-smoker mothers of same age and weight, with a risk of error less than 5% (p -value = 0.012). At same age and mother weight, child weight at birth is 267.213 g less for a smoker mother than for a non-smoker mother.

Note

We could have got the same conclusions by looking at confidence intervals and checking whether zero lies in the interval. If zero does not lie in the confidence interval, then the variable has a significant contribution in the model, after adjusting for the other variables.

```
> confint(model2)
              2.5 %      97.5 %
(Intercept) 1769.504181 2955.93509
AGE          -12.486773  26.67319
LWT           1.380732  16.34007
as.factor(SMOKE) 1 -475.945996 -58.48000
```



14.3.3 Confidence and Prediction Intervals for a New Value

Suppose we wish to predict the weight at birth of a child whose mother is 23 years old, weighs 57kg and smokes. The function `predict()` gives a prediction, a prediction interval and a confidence interval for the mean weight of children whose mothers have these characteristics.

```
> newdata <- data.frame(AGE=23,LWT=57,SMOKE=1)
> predict(model2,newdata,interval="pred")
  fit     lwr      upr
1 2763.693 1355.943 4171.444
> predict(model2,newdata,interval="conf")
  fit     lwr      upr
1 2763.693 2600.914 2926.472
```

14.3.4 Testing a Linear Sub-hypothesis: Partial Fisher Test

Fisher's partial test is used to test the contribution of a subset of explanatory variables in a model which already includes other explanatory variables. For example, consider the following two models:

- Model 1: $BWT = \beta_0 + \beta_1 LWT + \epsilon$
- Model 2: $BWT = \beta_0 + \beta_1 LWT + \beta_2 AGE + \beta_3 SMOKE + \epsilon$

Fisher's test is used to test the joint contribution of variables `AGE` and `SMOKE` in model 2. The hypotheses of the test are $\mathcal{H}_0 : \beta_2 = \beta_3 = 0$ and \mathcal{H}_1 : at least one of the coefficients β_2 or β_3 is non-zero. The following instructions are used for this test:

```
> anova(model1,model2)
Analysis of Variance Table
```

```

Model 1: BWT ~ LWT
Model 2: BWT ~ AGE + LWT + as.factor(SMOKE)
      Res.Df   RSS Df Sum of Sq    F Pr(>F)
1     187 96468171
2     185 92935223  2   3532949 3.5164 0.03171 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

The p -value of the test ($\text{Pr}(>F)=0.03171$) indicates that at least one of the two variables AGE or SMOKE gives extra information to predict child weight at birth, when mother weight has already been taken into account.

Note

When comparing two nested models which only differ by one variable, Fisher's partial test is equivalent to the individual Student test.

```

> model3 <- lm(BWT~LWT+SMOKE)
> anova(model3,model2)
Analysis of Variance Table
Model 1: BWT ~ LWT + SMOKE
Model 2: BWT ~ AGE + LWT + as.factor(SMOKE)
      Res.Df   RSS Df Sum of Sq    F Pr(>F)
1     186 93191828
2     185 92935223  1   256606 0.5108 0.4757

```

We get the same p -value as in the individual Student test associated with variable AGE in model 2.

14.3.5 Qualitative Variables with More Than Two Modalities

Binary explanatory variables are not an issue, as shown in the connecting thread example with variable SMOKE. Using such a variable in a regression model boils down to comparing the mean of the response variable Y in the two groups defined by the binary qualitative variable. This comparison is said to be adjusted on the other explanatory variables, meaning that we estimate the conditional mean of Y, for fixed values of all the explanatory variables but SMOKE.

However, for a qualitative variable with more than two modalities, we need to introduce **dummy variables** to compare the mean of Y in the groups defined by the modalities of the qualitative explanatory variable. A dummy variable for a group or modality is a variable that takes the value 1 in the group and 0 in all other groups.

Connecting thread example: Child weight at birth as a function of mother weight and race.

Variable RACE is coded with three modalities: 1 for white, 2 for black and 3 for other. In this example, we can therefore define three dummy variables RACE1, RACE2 and RACE3.

Categories	RACE	RACE1	RACE2	RACE3
White	1	1	0	0
Black	2	0	1	0
Other	3	0	0	1

```
> RACE1 <- as.integer(RACE==1)
> RACE2 <- as.integer(RACE==2)
> RACE3 <- as.integer(RACE==3)
```

If we try to fit the following model

$$BWT = \mathbf{X}\boldsymbol{\beta} + \epsilon = \beta_0 \mathbf{1} + \beta_1 LWT + \beta_2 RACE1 + \beta_3 RACE2 + \beta_4 RACE3 + \epsilon, \quad (14.1)$$

a problem will occur, as can be seen in the output below:

```
> summary(lm(BWT~LWT+RACE1+RACE2+RACE3))
Call:
lm(formula = BWT ~ LWT + RACE1 + RACE2 + RACE3)
Residuals:
    Min      1Q  Median      3Q     Max 
-2094.9 -420.8   40.1  478.2 1928.4 
Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2245.097   226.805   9.899 < 2e-16 ***
LWT          10.267    3.856   2.662  0.00844 **  
RACE1        243.667   113.826   2.141  0.03361 *   
RACE2       -209.098   168.988  -1.237  0.21752    
RACE3           NA        NA       NA       NA      
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
Residual standard error: 702.7 on 185 degrees of freedom
Multiple R-squared:  0.08578,    Adjusted R-squared:  0.07095 
F-statistic: 5.786 on 3 and 185 DF,  p-value: 0.0008399
```

This comes from the fact that the design matrix \mathbf{X} is not of full rank:

```
> head(model.matrix(summary(
+   lm(BWT~LWT+RACE1+RACE2+RACE3)))) # First rows of the design
# matrix.
  (Intercept)      LWT RACE1 RACE2 RACE3
1 1 82.55380 0 1 0
2 1 70.30681 0 0 1
3 1 47.62719 1 0 0
4 1 48.98797 1 0 0
5 1 48.53438 1 0 0
6 1 56.24545 0 0 1
```

Indeed, the sum of the last three columns gives the first column: the model is called non-identifiable. The least squares estimator of $\boldsymbol{\beta}$ is then no more unique (even if predictions based on any least squares estimator of $\boldsymbol{\beta}$ will be unique).

Note

We could think about using the least squares estimator $\hat{\beta}^\dagger = \mathbf{X}^+ \mathbf{BWT}$, based on the (Moore–Penrose) generalized inverse \mathbf{X}^+ de \mathbf{X} , defined on page 323 from Chap. 10.

```
> mpinv(model.matrix(summary(
+ lm(BWT~LWT+RACE1+RACE2+RACE3))))%*%as.matrix(BWT)
[1]
[1,] 1692.46453
[2,] 10.26709
[3,] 796.29883
[4,] 343.53360
[5,] 552.63210
```



But the trouble is that it is difficult to interpret the coefficients of this vector since non-controllable linear constraints on its components were used (in the sense that two of them are expressed in terms of all others).

We can see that the model (14.1) can be rewritten as

$$\begin{aligned} \text{BWT} &= \beta_0 + \beta_1 \text{LWT} + \beta_2 \text{RACE1} + \beta_3 \text{RACE2} + \beta_4 \text{RACE3} + \epsilon \\ &= \beta_0 + \beta_1 \text{LWT} + \beta_2(1 - \text{RACE2} - \text{RACE3}) + \beta_3 \text{RACE2} + \beta_4 \text{RACE3} + \epsilon \\ &= \underbrace{(\beta_0 + \beta_2)}_{\gamma_0} + \underbrace{\beta_1}_{\gamma_1} \text{LWT} + \underbrace{(\beta_3 - \beta_2)}_{\gamma_2} \text{RACE2} + \underbrace{(\beta_4 - \beta_2)}_{\gamma_3} \text{RACE3} + \epsilon. \end{aligned}$$

In this latter form, the model is now identifiable (γ_i 's can be estimated). This is easily verified on the following output:

```
> summary(lm(BWT~LWT+RACE2+RACE3))$coeff
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2488.76336 241.863663 10.289943 6.348821e-20
LWT          10.26709   3.856339  2.662393 8.442125e-03
RACE2        -452.76523 157.481809 -2.875032 4.513529e-03
RACE3        -243.66673 113.825910 -2.140697 3.360769e-02
```

This consists in taking as a reference the group RACE = 1 (white race), i.e. to consider the linear regression model with only the dummy variables for the other groups (RACE2, RACE3).

To fit a model with qualitative covariates, you can use the function `factor()` in instruction `lm()`, as can be seen below:

```
> model4 <- lm(BWT~LWT+factor(RACE))
> summary(model4)$coeff
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2488.76336 241.863663 10.289943 6.348821e-20
LWT          10.26709   3.856339  2.662393 8.442125e-03
factor(RACE) 2 -452.76523 157.481809 -2.875032 4.513529e-03
factor(RACE) 3 -243.66673 113.825910 -2.140697 3.360769e-02
```

Note that in this output, R used as reference the group RACE = 1.

In this context, the estimation $\hat{\beta}_2 = \widehat{\beta_3 - \beta_2} = -452.765$ g represents the difference of child birth weight between black mothers (RACE=2) and white mothers (reference group), and this result is significantly different from zero (p -value = 0.00451) in a model adjusted for mother weight. Similarly, the difference in average weight at birth between group RACE = 3 and the reference group is $\widehat{\beta_4 - \beta_2} = -243.667$ g and is significantly different from zero (p -value = 0.03361), adjusting for mother weight.

Note

It is possible to change the reference class using the function `relevel()`.

```
> summary(lm(BWT~LWT+relevel(factor(RACE), ref=3)))$coeff
              Estimate Std. Error
(Intercept) 2245.09663 226.805111
LWT          10.26709  3.856339
relevel(factor(RACE), ref = 3)1 243.66673 113.825910
relevel(factor(RACE), ref = 3)2 -209.09850 168.988169
                                         t value    Pr(>|t|)
(Intercept) 9.898792 8.296639e-19
LWT          2.662393 8.442125e-03
relevel(factor(RACE), ref = 3)1 2.140697 3.360769e-02
relevel(factor(RACE), ref = 3)2 -1.237356 2.175232e-01
```



Note

To test the global contribution of variable RACE, use a partial Fisher test, as described in the previous section.

```
> anova(model1,model4)
Analysis of Variance Table
Model 1: BWT ~ LWT
Model 2: BWT ~ LWT + factor(RACE)
Res.Df   RSS Df Sum of Sq    F    Pr(>F)
1     187 96468171
2     185 91346474  2   5121697 5.1864 0.006434 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



Warning

To resume, never add to the model the dummy variable for the reference group. For a qualitative variable with p modalities, only use $p - 1$ dummy variables.



14.3.6 Interaction Between Variables

We say that there is interaction between two explanatory variables X_1 and X_2 if the association between one of the variables and the response variable Y is not the same depending on the values of the other variable. This corresponds to the notion of effect modification in epidemiology.

Suppose we are interested in the association between X_1 and Y and we wish to know whether the effect of X_1 on Y is different depending on the values of X_2 . We simply need to include in the model the variables X_1 and X_2 as well as a third variable X_3 defined as the product of X_1 and X_2 ($X_3 = X_1 \times X_2$, called the interaction term between X_1 and X_2). To make things clearer, suppose we wish to determine whether the effect of quantitative variable X_1 is modified by binary (0/1) variable X_2 . We then consider the model:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 \times X_2 + \epsilon.$$

In group $X_2 = 0$, the model is: $Y = \beta_0 + \beta_1 X_1 + \epsilon$ and the effect of X_1 is measured by β_1 . In group $X_2 = 1$, the model is $Y = (\beta_0 + \beta_2) + (\beta_1 + \beta_3)X_1 + \epsilon$ and the effect of X_1 is measured by $\beta_1 + \beta_3$.

There is interaction between X_1 and X_2 (or modification of the effect of X_1 by X_2) if the effect of X_1 is different in groups $X_2 = 0$ and $X_2 = 1$, i.e. if $\beta_3 \neq 0$. We therefore need to perform an individual Student test on β_3 ($H_0 : \beta_3 = 0$ versus $H_1 : \beta_3 \neq 0$). If we accept H_1 , we keep the interaction term in the model: there is modification of effect.

Connecting thread example: Modification of the effect of mother age on child weight at birth by mother's smoking status.

We first consider the model

$$\text{BWT} = \beta_0 + \beta_1 \text{AGE} + \beta_2 \text{SMOKE} + \epsilon.$$

```
> model5 <- lm(BWT~AGE+SMOKE)
```

This model assumes that the effect of AGE is the same for smoker and non-smoker mothers. We can propose a more flexible model, allowing for different effects in the two groups SMOKE=0 and SMOKE=1:

$$\text{BWT} = \beta_0 + \beta_1 \text{AGE} + \beta_2 \text{SMOKE} + \beta_3 \text{AGE} \times \text{SMOKE} + \epsilon$$

```
> model6 <- lm(BWT~AGE+SMOKE+AGE:SMOKE)
```

Tip

To introduce an interaction term between the two variables X_1 and X_2 in a linear model, simply type $X1:X2$. Note that $X1*X2$ in a formula corresponds to $X1+X2+X1:X2$. The previous call could have been replaced with

```
> model6 <- lm(BWT~AGE*SMOKE)
```



The difference between the two models can be visualized on the next two plots. The first plot shows two parallel lines; the slope represents the effect of AGE on BWT, which is the same for smokers and non-smokers (Fig. 14.7).

```
> co <- coef(model5); a0 <- co[1] ; a1 <- co[1]+co[3]
> b <- co[2]
> fSMOKE <- as.factor(SMOKE)
> plot(BWT~AGE, xlab="AGE in years", ylab=
+ "Weight at birth (g)", main=expression(BWT ~ " = "
+ beta[0]+beta[1]*AGE+beta[2]*SMOKE+epsilon),
+ col = c('blue','red')[fSMOKE], pch=c(1,18)[fSMOKE])
> abline(a=a0,b,col="blue") ; abline(a=a1,b,col="red")

> legend("bottomright",c ("SMOKE=0","SMOKE=1"),
+ col=c("red","blue"),lty=1)
```

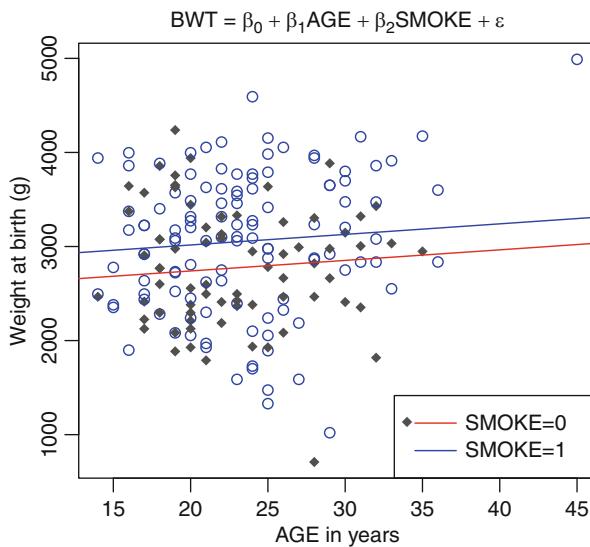


Fig. 14.7: Effect of age on BWT in a model without interaction

The next plot shows two different slopes, illustrating the notion of interaction (Fig. 14.8).

```

> co <- coef(model6); a0 <- co[1] ; a1 <- co[1]+co[3]
> b0 <- co[2]          # Effect of age on non-smokers.
> b1 <- co[2]+co[4]    # Effect of age on smokers.
> plot(BWT~AGE,xlab="AGE in years",ylab="Weight at birth (g)",
+ main=expression(BWT~"=~beta[0]+beta[1]*AGE+beta[2]*SMOKE+
+ beta[3]*AGE~"x"~SMOKE+epsilon))
> points(AGE[SMOKE==1],BWT[SMOKE==1],col="red",pch=18)
> points(AGE[SMOKE==0],BWT[SMOKE==0],col="blue")
> abline(a=a0,b0,col="blue") ; abline(a=a1,b1,col="red")

> legend("bottomright",c("SMOKE=0","SMOKE=1"),
+        col=c("red","blue"),lty=1)

```

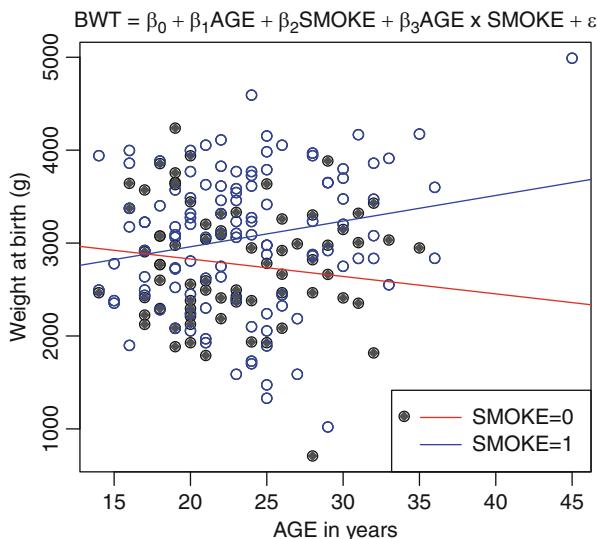


Fig. 14.8: Effect of age on BWT in a model with interaction

We can test the significance of the interaction term by analysing the results of model 6.

```

> summary(model6)
Call:
lm(formula = BWT ~ AGE * SMOKE)
Residuals:
    Min      1Q      Median      3Q      Max 
-2187.8  -456.6   52.8   526.6  1522.2 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2408.38     292.24   8.241 3.05e-14 ***
AGE          27.60      12.15   2.271   0.0243 *  
SMOKE       795.38     484.42   1.642   0.1023    

```

```

AGE:SMOKE      -46.36      20.45   -2.267    0.0245 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 709.4 on 185 degrees of freedom
Multiple R-squared:  0.0683,    Adjusted R-squared:  0.05319
F-statistic: 4.521 on 3 and 185 DF,  p-value: 0.004378

```

The coefficient β_3 is significantly different from zero (p -value = 0.024). We therefore conclude that the effect of mother age on child weight at birth is not the same depending on the smoking status of the mother. The results on association between mother age and child weight at birth must therefore be presented separately for the smoker and non-smoker groups.

For completeness sake, for non-smoker mothers, the effect of AGE is significant (p -value = 0.0243). The mean child weight at birth increases with mother age, by 27.60 g ($\hat{\beta}_1$) for one extra year for the mother. A confidence interval is given by

```

> confint(model16) [2,]
 2.5 % 97.5 %
3.628108 51.573048

```

For smoker mothers, the mean child weight at birth decreases with mother age, by 18.762 g ($\hat{\beta}_1 + \hat{\beta}_3$) for one extra year for the mother. To know whether this result is significant, we calculate the confidence interval for $\beta_1 + \beta_3$. A clever way of doing this is to create a new variable SMOKE1 which reverses the coding of SMOKE, then to launch the instructions

```

> SMOKE1 <- 1-SMOKE
> confint(lm(BWT~AGE+SMOKE1+AGE:SMOKE1)) [2,]
 2.5 % 97.5 %
-51.21423 13.68941

```

The value 0 lies in the confidence interval; thus the effect of AGE on BWT is not significant for smoker mothers.

14.3.7 Issues with Collinearity

When several explanatory variables give the same kind of information, several phenomena can occur:

- Disruption of estimate quality (very large variance)
- Contradictory values of coefficients (opposite signs)
- Non-significant coefficients

These are **collinearity issues**.

The criterion used to quantify collinearity of explanatory variables is the variance inflation factor VIF: $\frac{1}{1-r_j^2}$ where r_j^2 is the multiple coefficient of determination when we regress the j th explanatory variable x_j on the set of other regressors.

The VIF plays an essential role in estimator variance since $\text{Var}(\hat{\beta}_j | \mathbf{X} = \mathbf{x}) = \frac{\sigma^2}{n \times s_j^2} \times \frac{1}{1 - r_j^2}$ where $s_j^2 = \frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2$ is the variance of the sample x_j . The more x_j is collinear to the other regressors, the closer r_j^2 is to 1, hence the higher the term $\frac{1}{1 - r_j^2}$; the variance of the estimator $\hat{\beta}_j$ is then very high. Conversely, the closer r_j^2 is to 0, the closer the associated VIF is to 1 (the minimum). Thus, the more x_j is “independent” of the other regressors, the less the estimates are disrupted. Collinearity of regressors necessarily has an impact on estimator precision. We say there is strong collinearity when $\text{VIF}_j > 10$ (i.e. $r_j^2 > 0.9$).

The R instruction to compute the VIF is `vif()`, available in package `car`.

We show how to use the function `vif()` in the very simple example of model

$$\mathbb{E}[\text{BWT} | \text{LWT}, \text{AGE}] = \beta_0 + \beta_1 \text{LWT} + \beta_2 \text{AGE}.$$

```
> model17 <- lm(BWT ~ LWT + AGE)
> vif(model17)
      LWT          AGE
1.033513 1.033513
```

Note

In this case, the VIFs are equal, since there are only two regressors. This is a very simple example: there are only two regressors and we could have analysed this collinearity graphically, but the usefulness of this method is clear when there is a large number of regressors.

14.3.8 Variable Selection

Amongst the large number of possible explanatory variables, we wish to select those which explain Y the best. This way, we can decrease the number of predictors (giving a parsimonious model) and get good predictive power by eliminating redundant variables which increase the variance inflation factor (VIF).

The greater the number of parameters (large number of explanatory variables), the better the fit to the data (r^2 close to 1). As a trade-off, parameter estimation is disrupted (variance of estimators increases) because of collinearity issues.

In this section, we briefly present a few methods for variable selection available in R. They are illustrated on a few variables from data set `BIRTH-WEIGHT`.

We consider the explanatory variables LWT, AGE, UI, SMOKE and HT and two recoded variables FTV1 and PTL1. We note $FVT1 = 1$ if there was at least one visit to a physician, and $FVT1 = 0$ otherwise. Similarly, we note $PTL1 = 1$ if there is at least one preterm birth in the family history, and $PTL1 = 0$ otherwise.

► Best Subset Method

When the number p of explanatory variables is not too large, we can study all possibilities. One efficient algorithm (see [18, 19]) can go up to 30 variables: the leaps and bounds method. At fixed p , we choose the regression model with the largest r^2 . For two regression models with different numbers of explanatory variables, we can choose the one with largest adjusted r_a^2 .

The relevant R function is `leaps()` available in package `leaps()`.

```
> FVT1 <- FVT; FVT1 <- as.integer(FVT>=1)
> PTL1 <- PTL; PTL1 <- as.integer(PTL>=1)
> matx <- model.matrix(lm(BWT~ -1+LWT+AGE+UI+SMOKE+HT+FVT1+PTL1))
> # Equivalent to: matx <- cbind(LWT,AGE,UI,SMOKE,HT,FVT1,PTL1)
> adjr2.leaps <- leaps(matx,BWT,nbest=1,method="adjr2")
> best.model.adjr2 <- adjr2.leaps$which[adjr2.leaps$adjr2 ==
+                               max(adjr2.leaps$adjr2),]
> best.model.adjr2
  1      2      3      4      5      6      7
TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
```

The best model in the sense of the adjusted r_a^2 is thus

$$BWT = \beta_0 + \beta_1 LWT + \beta_2 UI + \beta_3 SMOKE + \beta_4 HT + \beta_5 PTL1 + \epsilon.$$

Note

You can use other selection criteria than the adjusted r_a^2 with the argument `method` of function `leaps()`. For example, `method="Cp"` uses Mallows' C_p criterion [27].



Another interesting function in package `leaps()` is `regsubsets()`. For example, with its argument `force.in`, it can be used to specify one or several variables which must be included in all models considered. We give an example using the BIC (Bayesian information criterion) to choose the best model [37] (Fig. 14.9):

```
> # Force SMOKE to be included:
> best.model.bic <- regsubsets(matzx,BWT,nbest=1,force.in=4)
> summary(best.model.bic)$bic # BIC values of best models of
# each size (nbest=1).
[1] -6.273748 -7.607040 -9.796126 -7.865648 -3.491572  1.544854
> plot(best.model.bic)
```

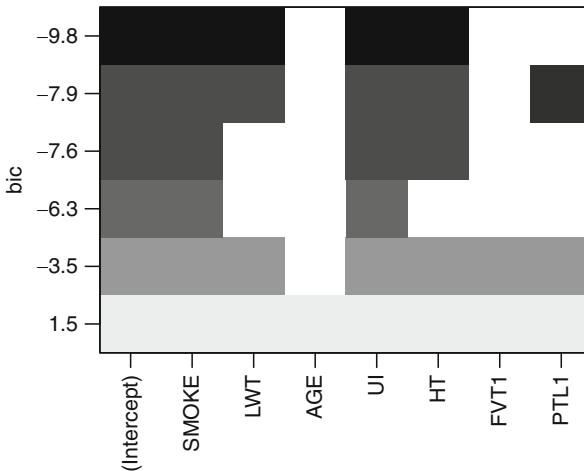


Fig. 14.9: Selecting variables with the BIC

The best model is the one with lowest BIC. The best model, in the sense of the BIC, is

$$BWT = \beta_0 + \beta_1 LWT + \beta_2 UI + \beta_3 SMOKE + \beta_4 HT + \epsilon.$$

Note

Other selection criteria can be used with argument `scale` in function `plot()` applied to an object of class `regsubsets` or using `$rsq`, `$rss`, `$adjr2` and `$cp` (for r -squared, residual sum of squares, adjusted r -squared and Mallows' Cp respectively), instead of `$bic`, with the function `summary()` as shown above.



► Forward Selection

The forward selection method is an iterative method. At each step, it selects the most significant explanatory variable (at level α) when we regress Y on all explanatory variables selected at previous steps and the newly chosen variable, as long as the marginal contribution of the new variable is significant.

Watch this method in action with function `add1()` for level $\alpha = 0.05$.

```
> add1(lm(BWT~1), BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1, test="F")
Single term additions
Model:
BWT ~ 1
  Df Sum of Sq    RSS    AIC F value    Pr(F)
<none>          99917053 2492.7
LWT   1  3448881 96468171 2488.0  6.6855  0.010481 *
AGE   1  806927  99110126 2493.1  1.5225  0.218790
UI    1  8028747  91888305 2478.8 16.3391 0.00007732 ***
SMOKE 1  3573406  96343646 2487.8  6.9359  0.009156 **
HT    1  2132014  97785038 2490.6  4.0772  0.044894 *
FVT1  1  1338322  98578731 2492.1  2.5387  0.112772
PTL1  1  4757523  95159530 2485.4  9.3491  0.002558 **

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

UI is the most significant variable.

```
> add1(lm(BWT~UI), BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1, test="F")
Single term additions
Model:
BWT ~ UI
  Df Sum of Sq    RSS    AIC F value    Pr(F)
<none>          91888305 2478.8
LWT   1  2076990  89811315 2476.5  4.3015  0.03946 *
AGE   1  472355  91415950 2479.9  0.9611  0.32819
SMOKE 1  2949940  88938365 2474.7  6.1693  0.01388 *
HT    1  3162469  88725836 2474.2  6.6296  0.01081 *
FVT1  1  949028  90939278 2478.9  1.9411  0.16522
PTL1  1  2837049  89051257 2474.9  5.9257  0.01587 *

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

HT is the most significant variable.

```
> add1(lm(BWT~UI+HT), BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1, test="F")
Single term additions
Model:
BWT ~ UI + HT
  Df Sum of Sq    RSS    AIC F value    Pr(F)
<none>          88725836 2474.2
LWT   1  3560080  85165756 2468.5  7.7333  0.005982 **
AGE   1  415275  88310561 2475.3  0.8700  0.352184
SMOKE 1  2828310  85897527 2470.1  6.0914  0.014492 *
FVT1  1  698035  88027801 2474.7  1.4670  0.227365
PTL1  1  2682800  86043036 2470.4  5.7683  0.017308 *

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

LWT is the most significant variable.

```
> add1(lm(BWT~UI+HT+LWT), BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,
+       test="F")
Single term additions
Model:
BWT ~ UI + HT + LWT
```

```

      Df Sum of Sq    RSS     AIC F value    Pr(F)
<none>           85165756 2468.5
AGE      1    94703  85071053 2470.3   0.2048  0.65138
SMOKE    1   2579898  82585858 2464.7   5.7480  0.01751 *
FVT1     1   509265  84656491 2469.3   1.1069  0.29414
PTL1     1   2127921  83037835 2465.7   4.7152  0.03118 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

SMOKE is the most significant variable.

```

> add1(lm(BWT~UI+HT+LWT+SMOKE), BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,
+       test="F")
Single term additions
Model:
BWT ~ UI + HT + LWT + SMOKE
      Df Sum of Sq    RSS     AIC F value    Pr(F)
<none>           82585858 2464.7
AGE      1    65305  82520553 2466.5   0.1448  0.70397
FVT1     1   275436  82310423 2466.0   0.6124  0.43491
PTL1     1   1434298  81151560 2463.3   3.2344  0.07375 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

No further variable is significant. The method thus stops at the model with variables: UI, HT, LWT and SMOKE.

► Backward Selection

This time, we start with the complete model and at each step, we delete the variable with lowest value of Student's test statistic (largest p -value) in absolute value, as long as it is not significant (at a specified level α).

Watch this method in action with function drop1() for level $\alpha = 0.05$.

```

> drop1(lm(BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1), test="F")
Single term deletions
Model:
BWT ~ LWT + AGE + UI + SMOKE + HT + FVT1 + PTL1
      Df Sum of Sq    RSS     AIC F value    Pr(F)
<none>           80692151 2466.3
LWT      1   2475214  83167365 2470.0   5.5521  0.0195277 *
AGE      1    87708  80779859 2464.5   0.1967  0.6578974
UI       1   5431112  86123263 2476.6  12.1825  0.0006059 ***
SMOKE    1   1622617  82314768 2468.0   3.6397  0.0580009 .
HT       1   3885141  84577292 2473.2   8.7147  0.0035749 **
FVT1     1   272400  80964551 2464.9   0.6110  0.4354262
PTL1     1   1601044  82293195 2468.0   3.5913  0.0596772 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

We delete variable AGE.

```
> drop1(lm(BWT~LWT+UI+SMOKE+HT+FVT1+PTL1), test="F")
Single term deletions
Model:
BWT ~ LWT + UI + SMOKE + HT + FVT1 + PTL1
      Df Sum of Sq    RSS    AIC F value    Pr(F)
<none>          80779859 2464.5
LWT     1   2740905  83520764 2468.8  6.1754 0.0138569 *
UI      1   5536620  86316478 2475.0 12.4742 0.0005228 ***
SMOKE   1   1644322  82424180 2466.3  3.7047 0.0558183 .
HT      1   3954174  84734033 2471.5  8.9089 0.0032279 **
FVT1    1   371701  81151560 2463.3  0.8375 0.3613362
PTL1    1   1530564  82310423 2466.0  3.4484 0.0649284 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We delete variable FVT1.

```
> drop1(lm(BWT~LWT+UI+SMOKE+HT+PTL1), test="F")
Single term deletions
Model:
BWT ~ LWT + UI + SMOKE + HT + PTL1
      Df Sum of Sq    RSS    AIC F value    Pr(F)
<none>          81151560 2463.3
LWT     1   2891443  84043002 2468.0  6.5203 0.0114803 *
UI      1   5763232  86914792 2474.3 12.9963 0.0004023 ***
SMOKE   1   1886275  83037835 2465.7  4.2536 0.0405794 *
HT      1   4217585  85369145 2470.9  9.5108 0.0023592 **
PTL1    1   1434298  82585858 2464.7  3.2344 0.0737548 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We delete variable PTL1.

```
> drop1(lm(BWT~LWT+UI+SMOKE+HT), test="F")
Single term deletions
Model:
BWT ~ LWT + UI + SMOKE + HT
      Df Sum of Sq    RSS    AIC F value    Pr(F)
<none>          82585858 2464.7
LWT     1   3311668  85897527 2470.1  7.3783 0.0072310 **
UI      1   6955671  89541530 2477.9 15.4971 0.0001171 ***
SMOKE   1   2579898  85165756 2468.5  5.7480 0.0175082 *
HT      1   4443587  87029445 2472.6  9.9002 0.0019278 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The method stops at the model with variables: UI, HT, LWT and SMOKE.

► The Stepwise Method

This algorithm is an improvement upon the forward selection method. At each step, it performs Student's or Fisher's tests, or optimize some criterion, to not add a non-significant variable and possibly delete previously included variables which

are no longer significant given the latest included variable. The algorithm stops when no variables can be added or deleted.

We present function `step()` to perform the stepwise method, using at each step a selection procedure based on the AIC criterion [2] (an information criterion). One can also use the well-known BIC criterion (Bayesian information criterion) by means of the `k=log(n)` argument of the `step()` function.

```
> step(lm(BWT~1), BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,
+       direction="both")
Start:  AIC=2492.66
BWT ~ 1
      Df Sum of Sq    RSS    AIC
+ UI     1   8028747 91888305 2478.8
+ PTL1   1   4757523 95159530 2485.4
+ SMOKE  1   3573406 96343646 2487.8
+ LWT    1   3448881 96468171 2488.0
+ HT     1   2132014 97785038 2490.6
+ FVT1   1   1338322 98578731 2492.1
<none>          99917053 2492.7
+ AGE    1   806927 99110126 2493.1
Step:  AIC=2478.83
BWT ~ UI
      Df Sum of Sq    RSS    AIC
+ HT     1   3162469 88725836 2474.2
+ SMOKE  1   2949940 88938365 2474.7
+ PTL1   1   2837049 89051257 2474.9
+ LWT    1   2076990 89811315 2476.5
<none>          91888305 2478.8
+ FVT1   1   949028 90939278 2478.9
+ AGE    1   472355 91415950 2479.9
- UI     1   8028747 99917053 2492.7
Step:  AIC=2474.21
BWT ~ UI + HT
      Df Sum of Sq    RSS    AIC
+ LWT    1   3560080 85165756 2468.5
+ SMOKE  1   2828310 85897527 2470.1
+ PTL1   1   2682800 86043036 2470.4
<none>          88725836 2474.2
+ FVT1   1   698035 88027801 2474.7
+ AGE    1   415275 88310561 2475.3
- HT     1   3162469 91888305 2478.8
- UI     1   9059202 97785038 2490.6
Step:  AIC=2468.47
BWT ~ UI + HT + LWT
      Df Sum of Sq    RSS    AIC
+ SMOKE  1   2579898 82585858 2464.7
+ PTL1   1   2127921 83037835 2465.7
<none>          85165756 2468.5
+ FVT1   1   509265 84656491 2469.3
+ AGE    1   94703 85071053 2470.3
- LWT    1   3560080 88725836 2474.2
- HT     1   4645559 89811315 2476.5
```

```

- UI      1  7482463 92648219 2482.4
Step: AIC=2464.66
BWT ~ UI + HT + LWT + SMOKE
   Df Sum of Sq    RSS    AIC
+ PTL1    1  1434298 81151560 2463.3
<none>           82585858 2464.7
+ FVT1    1  275436 82310423 2466.0
+ AGE     1   65305 82520553 2466.5
- SMOKE   1  2579898 85165756 2468.5
- LWT     1  3311668 85897527 2470.1
- HT      1  4443587 87029445 2472.6
- UI      1  6955671 89541530 2477.9
Step: AIC=2463.35
BWT ~ UI + HT + LWT + SMOKE + PTL1
   Df Sum of Sq    RSS    AIC
<none>           81151560 2463.3
+ FVT1    1  371701 80779859 2464.5
- PTL1    1  1434298 82585858 2464.7
+ AGE     1  187009 80964551 2464.9
- SMOKE   1  1886275 83037835 2465.7
- LWT     1  2891443 84043002 2468.0
- HT      1  4217585 85369145 2470.9
- UI      1  5763232 86914792 2474.3
Call:
lm(formula = BWT ~ UI + HT + LWT + SMOKE + PTL1)
Coefficients:
(Intercept)          UI            HT            LWT
  2631.45        -506.76       -633.15        9.33
   SMOKE          PTL1
  -208.46        -247.66

```

Warning

The data set BIRTH-WEIGHT has been used here only to illustrate how to use R functions for automatic selection, even though the best strategy for this data set would have been to select “by hand”.

Indeed, it should be noted that different methods of automatic selection may not lead to the same choice of variables in the final model. They have the advantage of being easy to use and of treating the question of variable selection in a systematic manner. The main drawback is that variables are included or deleted based on purely statistical criteria, without taking into account the aim of the study. This usually leads to a model which may be satisfactory from a statistical point of view, but in which the variables are not the most relevant when it comes to understanding and interpreting the data in the study.



14.3.9 Analysis of Residuals

We present here a few elements on analysis of residuals. This method is useful to check the assumptions of the model and to detect possible outliers. For further details, we recommend you read [41].

► Validating Model Assumptions

We have already mentioned that for simple linear regression, analysis of residuals can be used to check the assumptions of the regression model. We showed two plots to check the assumptions of normally distributed errors and homoscedasticity of errors (Figs. 14.10, 14.11).

Connecting thread example: Study “Child weight at birth”.

We study the validity of the assumptions for the model

$$\begin{aligned} \text{BWT} = & \beta_0 + \beta_1 \text{SMOKE} + \beta_2 \text{AGE} + \beta_3 \text{LWT} + \beta_4 \text{RACE2} + \beta_5 \text{RACE3} + \beta_6 \text{UI} \\ & + \beta_7 \text{HT} + \beta_8 \text{SMOKE} \times \text{AGE} + \epsilon. \end{aligned}$$

```
> finalmodel<-lm(BWT~SMOKE+AGE+LWT+factor(RACE)+UI+HT+SMOKE:AGE)
> par(mfrow=c(1:2))
> plot(finalmodel,1:2,col.smooth="red")
```

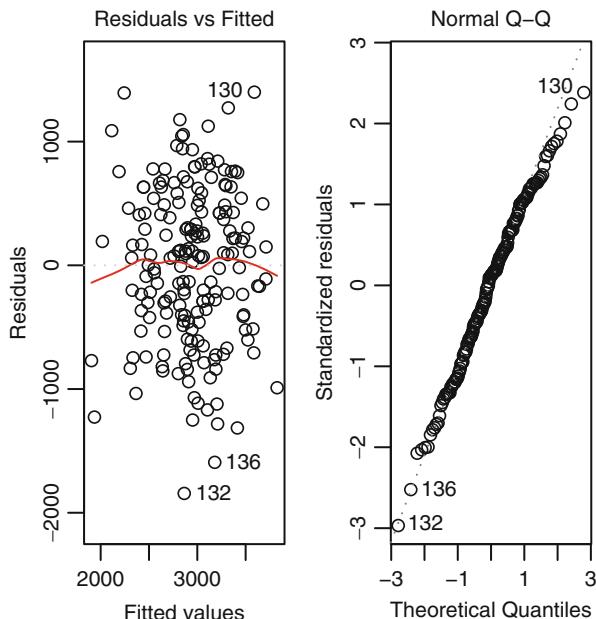


Fig. 14.10: Checking the assumptions of homoscedasticity (left) and normality (right)

```
> res <- residuals(finalmodel)
> par(mfrow=c(2,3))
> plot(res~SMOKE);plot(res~AGE);plot(res~LWT)
> plot(res~RACE);plot(res~UI);plot(res~HT)
```

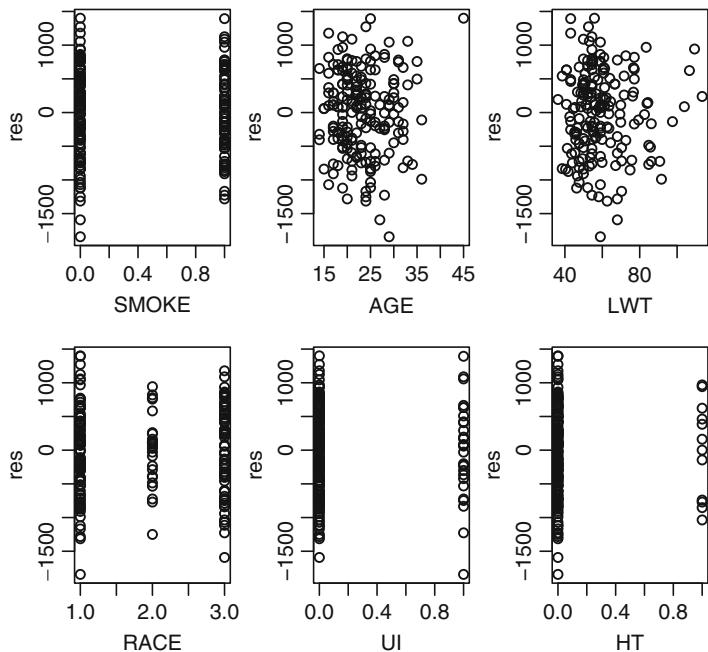


Fig. 14.11: Residuals as a function of explanatory variables

It can also be useful to plot the residuals as a function of each explanatory variable, as shown in the above figure. This plot is useful to check whether there is a relationship between the error term and the explanatory variables, which would invalidate the assumption of independence between errors and explanatory variables. This plot is also useful to detect outliers.

► Outliers and Influential Points

An outlier is a point with a large value of its residual. It is a point which is often distant from the others. It can be visualized on the plot of residuals against predicted values (or against an explanatory variable) as a point very far away. Several kinds of residuals can then be defined:

- **Standardized residuals** $t_i = \frac{\hat{\epsilon}_i}{\hat{\sigma} \sqrt{1 - h_{ii}}}$ where h_{ii} is the “leverage” (defined later on). These residuals are given by function `rstandard()`. Standardized residuals are “mostly” between -2 and 2 , but they are dependent.

- **Studentized residuals** $t_i^* = \frac{\hat{\epsilon}_i}{\hat{\sigma}_{(-i)} \sqrt{1 - h_{ii}}} = t_i \sqrt{\frac{n - p - 2}{n - p - 1 - t_i^2}}$ where $\hat{\sigma}_{(-i)}$ is an estimation of the standard deviation obtained without observation i . Studentized residuals are given by function `rstudent()`. An observation is called an outlier when $|t_i^*| > t_{0.975}^{(n-p-2)}$ (where $t_{0.975}^{(n-p-2)}$ is the 0.975 quantile of a Student distribution with $n - p - 2$ degrees of freedom) (Fig. 14.12):

```
> res.stud <- rstudent(finalmodel)      # Calculating studentized
                                         # residuals.
> threshold.stud <- qt(0.975,189-8-2) # Calculating the
                                         # threshold for the
                                         # Student distribution.
> cond <- res.stud<-threshold.stud | res.stud > threshold.stud
> # List of individuals who can be considered outliers.
> id.student <- ID[cond] # ID (identification number) is in
                           # first column of dataset.
> val.ajust <- fitted(finalmodel)
> plot(res.stud~val.ajust,xlab="Fitted values",
+       ylab="Studentized residuals",pch=20)
> abline(h=c(-threshold.stud,threshold.stud))

> text(val.ajust[cond],res.stud[cond],id.student,col="red",pos=1)
```

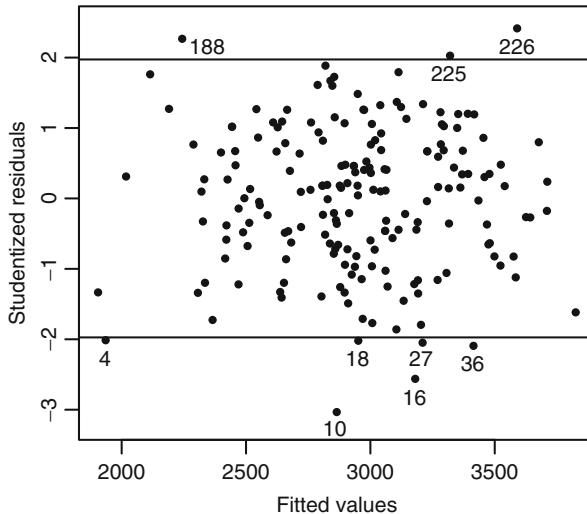


Fig. 14.12: Visualizing outliers: studentized residuals against fitted values

Another way of studying outliers is the notion of “leverage points”. The leverage for observation i (noted h_{ii}) is the value on the diagonal of matrix $\mathcal{H} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ (the hat matrix). This measure is mostly used for the variance of

residuals: $\text{Var}(\hat{\epsilon}_i) = \sigma^2(1 - h_{ii})$. A leverage above $2(p + 1)/n$ can be considered too large. Large h_{ii} indicates that the i th observation is far from the centre of gravity.

The h_{ii} , stored in vector leverage, are given by

```
> # Equivalent to hat(model.matrix(finalmodel)):  
> leverage <- hatvalues(finalmodel)  
> # We also could have used the following instructions:  
> outl <- influence.measures(finalmodel)  
> leverage <- outl$infmat[, "hat"]
```

To detect leverage points, you can type:

```
> threshold.leverage <- 2*(8+1)/189  
> outl.leverage <- ID[leverage>threshold.leverage]  
> # List of individuals with large leverage:  
> outl.leverage  
[1] 85 98 119 126 138 159 168 187 197 202 226 11 13 19  
[15] 20 28 75 83 84
```

Other diagnostics can be used to inspect outliers and check their influence on the regression model:

- **Cook's distance.** It is used to measure the influence of observation i on the estimation of the regression parameters. It is defined as

$$\begin{aligned} C_i &= \frac{\sum_{j=1}^n (\hat{y}_j - \hat{y}_j^{(-i)})^2}{\hat{\sigma}^2(p+1)} = \left(\frac{1}{p+1}\right) \left(\frac{h_{ii}}{1-h_{ii}}\right) t_i^2 \\ &= \left(\frac{1}{p+1}\right) \left(\frac{h_{ii}}{(1-h_{ii})^2}\right) \frac{\hat{\epsilon}_i^2}{\hat{\sigma}^2} \\ &= \left(\frac{1}{p+1}\right) \left(\frac{h_{ii}}{1-h_{ii}}\right) \frac{\hat{\sigma}_{(-i)}^2}{\hat{\sigma}^2} t_i^{*2}, \end{aligned}$$

where $\hat{y}_j^{(-i)}$ is the prediction at point $\mathbf{x}_j = (1, x_{j1}, \dots, x_{jp})^\top$ given by the model estimated without the i th observation.

A large value of C_i indicates that the i th observation is influential (1 is sometimes used as the threshold). Deleting this observation can lead to big modifications in the regression equation. The function to calculate Cook's distance is `cooks.distance()`.

Here is a graphical representation (Fig. 14.13):

```
> plot(cooks.distance(finalmodel), type="h")
> # Or: plot(finalmodel, 4)
```

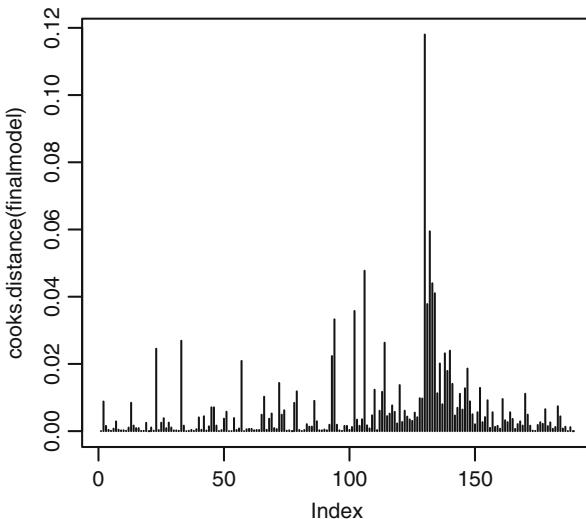


Fig. 14.13: Visualizing influential observations: Cook's distance

According to Cook's distance, no value seems globally influential.

- **Welsch–Kuh distance or Dffits.** It is defined as

$$\text{Dffits}_i = \frac{\hat{y}_i - \hat{y}_i^{(-i)}}{\hat{\sigma}_{(-i)} \sqrt{h_{ii}}} = t_i^* \sqrt{\frac{h_{ii}}{1 - h_{ii}}}.$$

Large $|\text{Dffits}_i|$ indicates that observation i has an influence on the estimate \hat{y}_i , which means that observation is influential on the regression results. In practice, an observation is considered influential if $|\text{Dffits}_i| \geq 2 \sqrt{\frac{p+1}{n}}$.

The R function to calculate the Dffits is `dffits()`.

```
> threshold.dffit <- 2*sqrt((8+1)/189)
> ID[abs(dffits(finalmodel))>=threshold.dffit]
[1] 108 119 187 188 197 202 210 226    4   10  11  13  18  20
```

- **The Dfbetas measure.** It is defined as

$$\text{Dfbetas}_{j,i} = \frac{\hat{\beta}_j - \hat{\beta}_j^{(-i)}}{\hat{\sigma}_{(-i)} \sqrt{(\mathbf{X}^\top \mathbf{X})_{j+1;j+1}^{-1}}}$$

where $\hat{\beta}_j^{(-i)}$ is the estimate of β_j obtained without using the i th observation. This quantity measures the influence of observation i on the estimate of the j th coefficient. For small or medium-size data sets, a value larger than 1 is suspect. For large data sets, observation i is suspect if $|\text{Dfbetas}_{j,i}| > 2/\sqrt{n}$ for at least one j .

The R function to calculate the Dfbetas is `dfbetas()`.

```
> threshold.dfbetas <- 1
> ID[apply(abs(dfbetas(finalmodel))>threshold.dfbetas,
+         FUN=any,MARGIN=1)]
integer(0)
```

Here, no value seems suspect.

- **Covariance ratio.** It is defined, for the i th observation, as the ratio of the determinant of the estimated matrix of variances-covariances of $\hat{\beta}_{(-i)}$ (estimator of β obtained without using the i th observation) by the determinant of the estimated matrix of variances-covariances of $\hat{\beta}$ (estimator of β):

$$\frac{\det \left[\widehat{\text{Var}}(\hat{\beta}_{(-i)}) \right]}{\det \left[\widehat{\text{Var}}(\hat{\beta}) \right]}.$$

If the ratio is near 1, the i th observation does not significantly influence the covariance matrix.

```
> max(abs(covratio(finalmodel)-1))
[1] 0.2897528
```

Note

For further details on these diagnostics, you can read [4], [41] or [11].



14.3.10 Polynomial Regression

In a polynomial model, the relationship between the explained variable Y and an explanatory variable X is represented in a non-linear fashion, such as:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_p X^p + \epsilon.$$

This model is a multiple regression model with p regressors: the powers of the explanatory variable.

To perform regression in a polynomial model, you simply need to correctly specify the formula associated with the model in the function `lm()`. Two R functions are useful: `I()` and `poly()`. The next table gives a few examples of formulae for polynomial models.

Model	R formula
$M_1: Y = \beta_0 + \beta_1 X + \beta_2 X^2$	$Y \sim \text{poly}(X, 2, \text{raw}=TRUE)$
$M_2: Y = \beta_1 X + \beta_2 X^2 + \beta_3 X^3$	$Y \sim -1 + \text{poly}(X, 3, \text{raw}=TRUE)$
$M_3: Y = \beta_0 + \beta_1 X + \beta_2 X^3$	$Y \sim X + I(X^3)$
$M_4: Y = \beta_1 X + \beta_2 X^3 + \beta_3 X^4$	$Y \sim -1 + X + I(X^3) + I(X^4)$

14.3.11 Summary

This table lists the main functions useful for multiple linear regression (Table 14.2).

Table 14.2: Main R functions for multiple linear regression

R instruction	Description
<code>pairs()</code>	Graphical inspection
<code>lm(Y ~ X1 + X2 + ... + X3)</code>	Estimation of the multiple linear model
<code>summary(lm())</code>	Description of the results of the model
<code>confint(lm())</code>	Confidence interval for regression parameters
<code>predict()</code>	Function for predictions
<code>plot(lm())</code>	Graphical analysis of residuals
<code>anova(mod1, mod2)</code>	Partial Fisher test
<code>X1:X2</code>	Interaction between X_1 and X_2
<code>vif()</code>	Computation of VIF

Memorandum

`lm()`: perform linear regression
`summary(lm())`: results of the linear model
`confint()`: confidence interval for regression parameters
`predict()`: prediction of new values
`residuals()`: recover residuals of a linear model
`plot(lm())`: plots for model validation
`pairs()`: scatter plot
`anova(lm())`: analysis of variance table of a linear model
`X1X2`: interaction term
`rstandard()`: standardized residuals
`rstudent()`: studentized residuals
`vif()`: calculation of VIF
`cooks.distance()`: Cook's distance
`dffits()`: Welsch–Kuh distance
`dfbetas()`: Dfbetas measure
`step()`: stepwise method with AIC
`regsubsets()`: selection by exhaustive search



Exercises

- 14.1-** Give the instruction to fit the model $Y = \beta_0 + \beta_1 X_1 + \epsilon$.
- 14.2-** Give the instruction to fit the model $Y = \beta_1 X_1 + \epsilon$.
- 14.3-** Give the instruction to fit the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$.
- 14.4-** Give the instruction to fit the model $Y = \beta_0 + \beta_1 X_1 \times \beta_2 X_2 + \beta_3 X_1 + \beta_4 X_2 + \epsilon$.
- 14.5-** Give the instruction to fit the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_1^4 + \epsilon$.
- 14.6-** Which instruction performs a partial Fisher test?
- 14.7-** Which function recovers the residuals of a model?
- 14.8-** Which function gives the estimates of a regression model?
- 14.9-** Let Z be a qualitative variable. Which function should you use to fit a regression model with Z as the explanatory variable?
- 14.10-** Give the instruction to fit the polynomial model $Y = \beta_0 + \beta_1 X^1 + \beta_2 X^2 + \beta_3 X^3 + \epsilon$.
- 14.11-** Which function performs forward selection?
- 14.12-** Which function performs backward selection?



Worksheet

A- Study of Simple Linear Regression

- *Study of synthetic data:*

14.1- Simulate a data set (x_i, y_i) , $i = 1, \dots, n$ from a simple linear regression model. To this end:

- Choose the true parameters β_0 and β_1 , as well as $\sigma > 0$.
- Simulate the vector of errors ϵ of size n from a normal distribution $\mathcal{N}(0, \sigma^2)$.
- Simulate the vector of values of the explanatory variable x of size n from a uniform distribution over $[0, t]$ where t is a positive real number of your choosing.
- Construct the vector of values of the explained variable y of size n from the linear regression model.

14.2- Plot the n points (x_i, y_i) .

14.3- Give an estimate of the regression parameters and of the error variance.

14.4- Analyse the residuals to validate the model:

- Plot the residuals against the fitted values.
- Plot the fitted values against the observed values.
- Draw a plot to check the normality of residuals.

14.5- Change the values of n and σ to understand the consequences on the precision of the regression parameter estimates (in terms of variance).

• *Study of intima media:* In the “Intima–media” study, we wish to study the relationship between intima–media thickness and age.

14.1- Download the intima-media data file.

14.2- Plot variable `measure` as a function of variable `AGE`. Describe this scatter plot.

14.3- Is there a link between these variables? Explain how to measure the severity of the link.

14.4- We now wish to fit a regression line on this scatter plot:

- Propose a regression model and estimate the parameters of the model.
- Draw the regression line over the scatter plot.

14.5- Analyse the residuals to validate the model.

14.6- Give a prediction interval for intima–media thickness for a 33-year-old person.

14.7- Give a confidence interval for the average intima–media thickness of a 33-year-old person.

- 14.8-** Propose a model to increase the predictive power of intima–media thickness, using only AGE as an explanatory variable.

B- Study of Multiple Linear Regression

- *Study of intima media:* In the previous practical, we looked at the relationship between intima–media thickness and age. We now wish to fit a regression model on all variables which may explain variations in intima–media thickness. The study will rely on the following variables: AGE, SPORT, alcohol, packyear and the variable BMI which you must create from variables height, weight.

We are mostly interested in tobacco, through the variable packyear as main exposure factor. We therefore decide to keep this variable in the model even if it is not significant.

- 14.1-** Draw scatter plots of all pairs of variables (explained and explanatory). Do you suspect any issues with collinearity?
- 14.2-** Perform a univariate analysis of intima–media thickness of each explanatory variable.
- 14.3-** We only keep the explanatory variables associated with significance level $p < 0.25$ in the univariate analysis. One by one, test all possible interactions between the selected explanatory variables and the main exposure variable packyear.
- 14.4-** Estimate and analyse the model with all variables which were declared significant in the univariate analysis ($\alpha = 25\%$) and all interaction terms significant at the 10 % level.
- 14.5-** Are the interaction terms still significant? Remove interaction terms which are no longer significant at the 10 % level.
- 14.6-** Starting with the model from the previous question, remove one by one all variables which are not significant at the 5 % level, making sure that the removals do not make a big difference on the estimate of the coefficient associated with tobacco status.
- 14.7-** Interpret the final model.

- *Study of unemployment rates:* This practically studies unemployment rates from 1960 to 1993. The data set **unemployment** is made of $n = 34$ yearly observations (from 1960 to 1993). Here is a description of the variables:

- year: year
- unemp: unemployment rate
- gdprate: rate of variation of gross domestic product (GDP), representing economic growth
- govspend: ratio of government spending and GDP, representing the degree of intervention of the state in the economy
- taxb: tax burden, to see whether taxation of businesses has an impact on hiring policy, and hence on unemployment rates

- **salav**: ratio of salaries to added value, to know the influence of cost at hiring
- **infl**: inflation rate, to verify the inverse relationship between unemployment and inflation, defined in the Philips curve

- 14.1-** We consider a linear model explaining variable **unemp** as a function of variable **gdprate** only. Download the data set <http://www.biostatisticien.eu/springeR/unemployment.RData>. Perform a complete analysis of the underlying simple linear regression model.
- 14.2-** We consider a multiple linear model explaining variable **unemp** as a function of all explanatory variables in the data set (except variable **year**). Give the correlation matrix of all these variables.
- 14.3-** Draw scatter plots of all pairs of variables.
- 14.4-** Which explanatory variables seem to make the biggest contribution? Do you suspect collinearity between regressors?
- 14.5-** Present the results of the multiple linear regression model with explanatory variables.
- 14.6-** Calculate the VIF associated with each explanatory variable.
- 14.7-** Perform backward variable selection with threshold $\alpha = 0.2$.
- 14.8-** Present the final model.
- 14.9-** Suppose we do not know the value of **unemp** in 1993. Can you predict its value, and calculate a 95 % prediction interval?
- 14.10-** What is the observed value of **unemp** in 1993? Is this surprising?

C- Study of Polynomial Regression

- *Study of synthetic data:*

- 14.1-** Simulate a sample of size $n = 100$ from the following model:

$$Y = X + 2X^2 + 3.5X^3 - 2.3X^4 + \epsilon,$$

where X follows a uniform distribution over $[-2, +2]$ and ϵ follows a $\mathcal{N}(0, 1)$ distribution.

- 14.2-** Draw the scatter plot and the simulated polynomial line.
- 14.3-** Fit a simple linear regression model. Remember to analyse the residuals.
- 14.4-** Fit a polynomial regression, using a polynomial of degree 4. Draw the estimated model over the scatter plot.

- *Fitting a scatter plot with a polynomial:* Suppose you are asked to propose a model to predict a variable Y given an explanatory variable X . You are given a sample of size n .

- 14.1-** Download the data file <http://www.biostatisticien.eu/springeR/fitpoly.RData>.
- 14.2-** Draw a scatter plot of variable Y as a function of variable X .

- 14.3-** Is there a linear relationship between these two variables? Fit a regression line on the previous plot.
- 14.4-** Perform polynomial regression to fit the data better.
- 14.5-** Draw the estimated polynomial over the scatter plot. Draw the confidence curve for the mean of Y for $X \in [-3.5, 3.5]$. Add the prediction interval of the model for $X \in [-3.5, 3.5]$.

Chapter 15

Elementary Analysis of Variance

Prerequisites and goals of this chapter

- Read Chap. 14.
- This chapter describes the various R commands to perform analysis of variance. We present the standard cases of analysis of variance with 1 factor and 2 factors with or without interaction. We also introduce repeated measures analysis of variance.

— SECTION 15.1 —

Analysis of Variance with One Factor

15.1.1 Aims, Data and Model

- **Aim:** Analysis of variance (ANOVA) is a method to study the modification of the mean μ of a phenomenon under study Y (quantitative variable) under the influence of one or several qualitative experimental factors (treatments). In the case where the mean is only influenced by one factor (noted X), it is called **analysis of variance with one factor** or **one-way ANOVA**. A factor is often a qualitative variable with a small number of modalities. The number of modalities (or levels) of factor X is noted I . We assume that Y follows a normal distribution $\mathcal{N}(\mu_i, \sigma^2)$ in each subpopulation i defined by the modalities of X. The aim is to test whether these I populations have equal means, i.e. to test the null hypothesis

$$\mathcal{H}_0 : \mu_1 = \mu_2 = \cdots = \mu_I$$

against the assertion of interest

$$\mathcal{H}_1 : \exists i \neq i' / \mu_i \neq \mu_{i'} \text{ ("there are at least two different means").}$$

- *Data:* For each subpopulation i (or modality i of X, or group i), we are given a sample of n_i observations from the quantitative variable Y:

$$y_{i,1}, y_{i,2}, \dots, y_{i,n_i}.$$

The model is written

$$Y_{ik} = \mu_i + \epsilon_{ik}, \quad \text{for } k = 1, \dots, n_i \text{ and } i = 1, \dots, I,$$

where the errors ϵ_{ik} are independent random variables following the distribution $\mathcal{N}(0, \sigma^2)$. We can also write μ_i as $\mu_i = \mu + \alpha_i$ for $i = 1, \dots, I$. In this setting, μ is called the *mean effect* of the factor and $\alpha_i = \mu_i - \mu$ is called the *differential effect for level i* of the factor. The above model can therefore also be written as

$$Y_{ik} = \mu + \alpha_i + \epsilon_{ik}, \quad \text{for } k = 1, \dots, n_i \text{ and } i = 1, \dots, I.$$

This model is not identifiable (which means that some parameters cannot be estimated). We therefore have to impose a (linear) constraint to make it identifiable, for example $\sum_{i=1}^I \alpha_i = 0$, which corresponds to taking the mean effect μ as the reference.

Warning

 By default, R imposes the constraint $\alpha_1 = 0$. All comparisons are then made relatively to μ_1 . The reference class is level 1 of the factor.

See also

 The concept of a reference group has already been explained on page 476 of Chap. 14.

15.1.2 Example and Graphical Inspection

- *Example of use:* Cold sores.

Five treatments (T_1, \dots, T_5) against cold sores, including one placebo, were randomly assigned to 30 patients (six patients per treatment group). For each patient, the time (in days) between the apparition of the cold sore and complete scarring was measured.

		Treatments				
		T_1 (placebo)	T_2	T_3	T_4	T_5
	5		4	6	7	9
	8		6	4	4	3
	7		6	4	6	5
	7		3	5	6	7
	10		5	4	3	7
	8		6	3	5	6

The aim here is to compare the theoretical means of scarring times; these times were observed on five independent samples (treatment groups.)

- *Graphical inspection.* We first perform a brief descriptive analysis of the data, to see whether any probable patterns emerge (Fig. 15.1).

```
> X <- data.frame(Placebo=c(5,8,7,7,10,8),T2=c(4,6,6,3,5,6),
+   T3=c(6,4,4,5,4,3),T4=c(7,4,6,6,3,5),T5=c(9,3,5,7,7,6))
> times <- stack(X)$values # stack() is used to stack vectors.
> treatment <- stack(X)$ind
> tapply(times,treatment,summary)
$Placebo
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  5.0    7.0    7.5   7.5    8.0   10.0
$T2
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  3.00   4.25   5.50   5.00   6.00   6.00
$T3
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  3.000  4.000  4.000  4.333  4.750  6.000
$T4
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  3.000  4.250  5.500  5.167  6.000  7.000
$T5
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  3.000  5.250  6.500  6.167  7.000  9.000
> plot(times~treatment)
```

15.1.3 ANOVA Table and Parameter Estimation

- *R instruction for the ANOVA table:* Use the function `aov()`. As for regression models, ANOVA works with R formulae. You therefore have to specify the model.

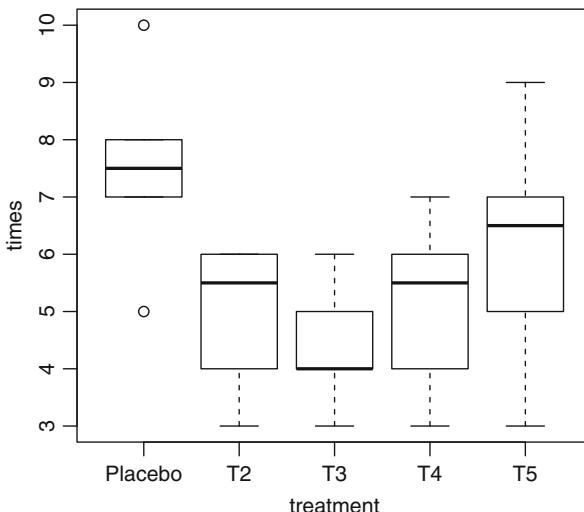


Fig. 15.1: Box plot of scarring times for each treatment

```
> my.aov <- aov(times~treatment)
> summary(my.aov)
      Df Sum Sq Mean Sq F value    Pr(>F)
treatment     4 36.467  9.1167   3.896 0.01359 *
Residuals   25 58.500   2.3400
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Warning

If it has not already been done, remember to declare your factor variable (in this case, the variable `treatment`) as an R object of type `factor`, using the function `factor()`.

```
> class(treatment)
[1] "factor"
```

ANOVA is in fact a linear model, so note that it is also possible to perform analysis of variance of the underlying linear model:

```
> model <- lm(times~treatment)
> anova(model)
Analysis of Variance Table
Response: times
      Df Sum Sq Mean Sq F value    Pr(>F)
treatment     4 36.467  9.1167   3.896 0.01359 *
Residuals   25 58.500   2.3400
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Tip

Note that you can also use the function `Anova()` from package `car`. This function is more complete and can handle more complex data.



The analysis of variance table outputs the result of Fisher's test for the hypotheses: $\mathcal{H}_0 : \mu_1 = \mu_2 = \dots = \mu_I$ and $\mathcal{H}_1 : \exists i \neq i' / \mu_i \neq \mu_{i'}$ ("there are at least two different means"). The p -value=0.013 allows us to conclude that the effects of at least two treatments are different, although we do not know which ones.

- *Estimating the parameters of the model:* Estimates are given by the function `summary()` for model `lm(times~treatment)`. Recall that R imposes the constraint $\alpha_1 = 0$.

```
> summary(model)
Call:
lm(formula = times ~ treatment)
Residuals:
    Min      1Q  Median      3Q     Max 
-3.16667 -0.87500 -0.08333  0.83333  2.83333 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  7.5000   0.6245 12.010 7.06e-12 ***
treatmentT2 -2.5000   0.8832 -2.831  0.00903 **  
treatmentT3 -3.1667   0.8832 -3.586  0.00142 **  
treatmentT4 -2.3333   0.8832 -2.642  0.01401 *   
treatmentT5 -1.3333   0.8832 -1.510  0.14366  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 1.53 on 25 degrees of freedom
Multiple R-squared:  0.384,      Adjusted R-squared:  0.2854 
F-statistic: 3.896 on 4 and 25 DF,  p-value: 0.01359
```

The intercept corresponds to the estimate of the mean time for the placebo (treatment 1 is the reference). The estimate associated with variable T2 corresponds to the differential effect between treatment T2 and the placebo. The same goes for the other variables. The assertions of interest and the two-by-two Student tests performed in this model (with constraint $\alpha_1 = 0$) are summed up in the following table:

	\mathcal{H}_1
Intercept	$\mu_1 \neq 0$
Treatment T2	$\alpha_2 \neq 0 \Leftrightarrow \mu_1 \neq \mu_2$
Treatment T3	$\alpha_3 \neq 0 \Leftrightarrow \mu_1 \neq \mu_3$
Treatment T4	$\alpha_4 \neq 0 \Leftrightarrow \mu_1 \neq \mu_4$
Treatment T5	$\alpha_5 \neq 0 \Leftrightarrow \mu_1 \neq \mu_5$

The results output by R show that there is a significant difference between the placebo and treatments 2, 3 and 4. In this setting, the placebo was the natural

reference. However, it is possible to take another reference, or a linear constraint, using the instruction `C()`, as shown in the next example:

```
> summary(lm(times~C(treatment,base=2)))
Call:
lm(formula = times ~ C(treatment, base = 2))
Residuals:
    Min      1Q  Median      3Q     Max 
-3.16667 -0.87500 -0.08333  0.83333  2.83333 
Coefficients:
            Estimate Std. Error t value
(Intercept) 5.00000   0.6245   8.006
C(treatment, base = 2)1 2.5000   0.8832   2.831
C(treatment, base = 2)3 -0.6667   0.8832  -0.755
C(treatment, base = 2)4  0.1667   0.8832   0.189
C(treatment, base = 2)5  1.1667   0.8832   1.321
Pr(>|t|)    
(Intercept) 0.0000000232 ***
C(treatment, base = 2)1 0.00903 ** 
C(treatment, base = 2)3 0.45739  
C(treatment, base = 2)4 0.85184  
C(treatment, base = 2)5 0.19847  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 1.53 on 25 degrees of freedom
Multiple R-squared:  0.384,    Adjusted R-squared:  0.2854 
F-statistic: 3.896 on 4 and 25 DF,  p-value: 0.01359
```

Fisher's statistic value does not change, since it does not depend on the linear constraint, but the estimates and the individual Student tests do change. With these results, we cannot show that treatment 2 is different to treatments 3, 4 and 5, but we do get a significant Student's test for the comparison of the placebo with treatment 2.

Note



To get the constraint $\sum_{i=1}^I \alpha_i = 0$, use `C(treatment, sum)`.

Note that you can use the function `model.matrix()` on the fitted linear model to get the matrix of explanatory variables. In the ANOVA model with a constraint of the type $\alpha_i = 0$, the matrix includes an intercept (a column of 1s) and the $I - 1$ indicator variables.

15.1.4 Validation of Assumptions

- **Validation of assumptions:** The ANOVA model corresponds to a linear model with a qualitative explanatory variable. The assumptions of the model can be validated using the method of analysis of residuals we presented for the regression model. Recall that the plot of residuals is obtained with the instructions (Fig. 15.2):

```
> par(mfrow=c(2,2))
> plot(model,col.smooth="red")
```

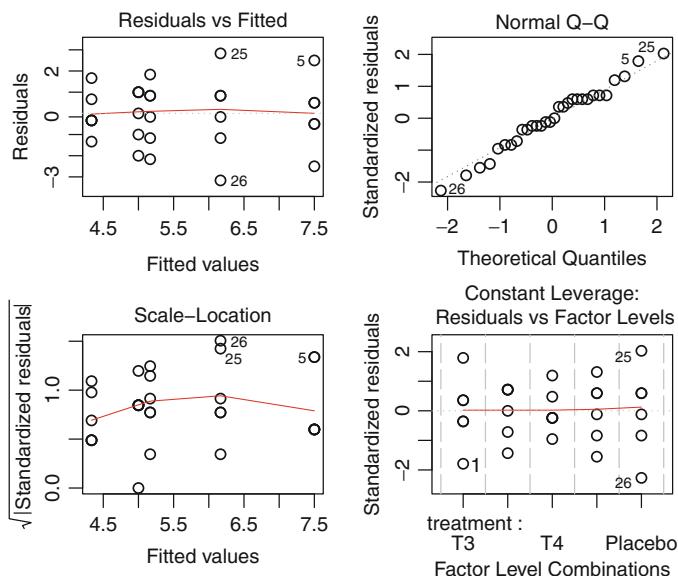


Fig. 15.2: Analysing the residuals in single-factor ANOVA

In ANOVA, it is also possible to use a test of equality of variances to explore whether the assumption of homoscedasticity is admissible. Bartlett's test (under normality in the subpopulations) can be obtained by

```
> bartlett.test(times~treatment)
Bartlett test of homogeneity of variances
data: times by treatment
Bartlett's K-squared = 2.4197, df = 4, p-value = 0.6591
```

However, this test is not robust to non-normality. In that case, use Levene's test [25]:

```
> levene.test(times,treatment) # Available in package car.
Levene's Test for Homogeneity of Variance (center = median)
 Df F value Pr(>F)
group  4   0.5851  0.6763
25
```

15.1.5 Multiple Comparisons and Contrasts

- **Multiple comparisons:** If, after performing analysis of variance, we reject the hypothesis of equal means relative to a factor with I levels, an interesting question is which means are significantly different from the others. In the cold sore example, we would like to select the most efficient treatment, i.e. the one which leads to the fastest scarring.

The individual Student test in the linear model is perfectly valid to compare two treatments chosen in advance. However, it cannot be used to compare (for example) the treatment which seems to give the best results with the treatment which seems to give the worse results. Indeed, this would be equivalent to comparing all pairs of treatments. Each test has probability α (the level of the test) of showing as present a difference which does not exist. Overall, out of the $I(I - 1)/2$ possible comparisons, the probability that one is declared significant “at random” becomes important. There are several methods to control the global risk for the $I(I - 1)/2$ pairwise comparisons.

The function `pairwise.t.test()` performs all pairwise comparisons and includes several methods to correct the risk α in order to take into account the problem of multiple tests.

```
> pairwise.t.test(times, treatment, p.adjust="bonf")
  Pairwise comparisons using t tests with pooled SD
data: times and treatment
Placebo T2      T3      T4
T2  0.090   -      -      -
T3  0.014  1.000   -      -
T4  0.140  1.000  1.000   -
T5  1.000  1.000  0.483  1.000
P value adjustment method: bonferroni
```

R gives p -values adjusted with Bonferroni’s correction: the corrected p -values are given by multiplying the p -values from the Student tests by the number of tests. Given the results (p -value = 0.014 between treatment 1 and treatment 3), there is a significant difference between treatment 1 (placebo) and treatment 3, at the 5 % level.

Note

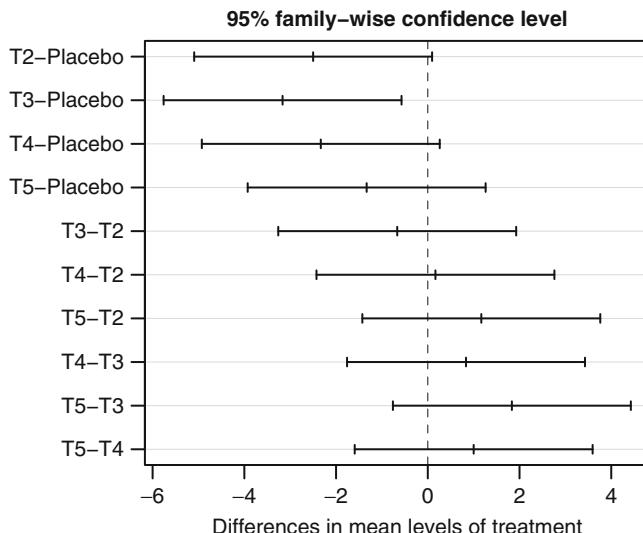
The comparison between treatment 1 and treatment 3 was performed by analysing model 1. The p -value for the individual Student test was 0.0014. Since ten comparisons were performed, this p -value is multiplied by 10 in Bonferroni’s method.



Many other correction methods exist. For single-factor analysis of variance with the same number of observations in each group, Tukey's method [30] is the most accurate. It gives simultaneous confidence intervals for the difference $\mu_i - \mu_j$ where $1 \leq i < j \leq I$.

```
> my.aov <- aov(times~treatment)
> TukeyHSD(my.aov)
  Tukey multiple comparisons of means
    95% family-wise confidence level
Fit: aov(formula = times ~ treatment)
$treatment
   diff      lwr      upr      p adj
T2-Placebo -2.5000000 -5.0937744  0.09377442 0.0627671
T3-Placebo -3.1666667 -5.7604411 -0.57289224 0.0113209
T4-Placebo -2.3333333 -4.9271078  0.26044109 0.0927171
T5-Placebo -1.3333333 -3.9271078  1.26044109 0.5660002
T3-T2      -0.6666667 -3.2604411  1.92710776 0.9410027
T4-T2      0.1666667 -2.4271078  2.76044109 0.9996956
T5-T2      1.1666667 -1.4271078  3.76044109 0.6811222
T4-T3      0.8333333 -1.7604411  3.42710776 0.8770466
T5-T3      1.8333333 -0.7604411  4.42710776 0.2614661
T5-T4      1.0000000 -1.5937744  3.59377442 0.7881333
```

```
> par(las=1) # Horizontal writing of labels.
> plot(TukeyHSD(my.aov))
```



The results from Tukey's method agree with those from Bonferroni's method: the only confidence interval which does not contain the value 0 is the one for the difference between treatment 3 and treatment 1. There is a significant difference between treatment 1 and treatment 3. Since scarring time is shorter with treatment 3, we suggest using treatment 3.

- *Contrast analysis:* In ANOVA, a contrast (noted L) is defined as a linear combination of theoretical means with sum of coefficients equal to zero:

$$L = \sum_{i=1}^I \lambda_i \mu_i = \boldsymbol{\lambda}^\top \boldsymbol{\mu} \quad \text{with} \quad \sum_i \lambda_i = 0,$$

where $\boldsymbol{\mu} = (\mu_1, \dots, \mu_I)^\top$ and $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_I)^\top$.

Contrasts are used to compare the means of groups of levels. For example, in the cold sore example, to compare treatments 2 and 3, you should use the contrast $L_1 = \boldsymbol{\lambda}^\top \boldsymbol{\mu}$ with $\boldsymbol{\lambda} = (0, 1, -1, 0, 0)^\top$ and perform the test $\mathcal{H}_0 : L_1 = 0$ against $\mathcal{H}_1 : L_1 \neq 0$.

You can perform tests on contrasts with the function `fit.contrast()` from package `gregmisc`.

```
> require("gregmisc")
> cmat <- rbind(": 2 against 3"=c(0,1,-1,0,0),
+                  "treatment: 2 against 3 0.6666667 0.883176 0.7548514
+                                         Pr(>/t/)")
+                  "treatment: 2 against 3 0.4573908"
```

Note



The same result was given by the individual Student test in model 2.

Now suppose that treatments 2 and 3 are ointments but treatments 4 and 5 are anti-tobacco patches. To compare the two types of remedies, you can use the contrast $L_2 = \boldsymbol{\lambda}^\top \boldsymbol{\mu}$ with $\boldsymbol{\lambda} = (0, -1, -1, 1, 1)^\top$ and perform the test $\mathcal{H}_0 : L_2 = 0$ against $\mathcal{H}_1 : L_2 \neq 0$.

```
> cmat <- rbind(": 2 against 3"=c(0,1,-1,0,0),
+                  "treatment: 2 and 3 against 4 and 5"=c(0,-1,-1,1,1))
+                  "treatment: 2 and 3 against 4 and 5 2.0000000 1.2490000
+                                         t value  Pr(>/t/)"
+                  "treatment: 2 against 3 0.7548514 0.4573908
+                  "treatment: 2 and 3 against 4 and 5 1.6012815 0.1218767"
```

15.1.6 Summary

The next table presents the main functions for single-factor analysis of variance (Table 15.1).

Table 15.1: Main functions for single-factor ANOVA

R instruction	Description
<code>plot(Y~factor(X))</code>	Graphical inspection
<code>aov(Y~factor(X))</code>	Analysis of variance
<code>summary(aov(Y~factor(X)))</code>	Analysis of variance table
<code>anova(lm(Y~factor(X)))</code>	Analysis of variance table
<code>pairwise.t.test()</code>	Pairwise comparisons
<code>fit.contrast()</code>	Contrast tests (package <code>gregmisc</code>)
<code>barlett.test()</code>	Homoscedasticity test
<code>levene.test()</code>	Homoscedasticity test
<code>plot(aov(Y~factor(X)))</code>	Graphical analysis of residuals

SECTION 15.2

Analysis of Variance with Two Factors**15.2.1 Aims, Data and Model**

- **Aim:** ANOVA with two factors, or two-way ANOVA, is a method to explain a quantitative variable with two “crossed” explanatory qualitative variables (called factors).

Note

In ANOVA, the explanatory variables are often called independent variables (e.g., in psychology) and noted IV.



- **Data:** Let A be a factor with I modalities and B a factor with J modalities. For each couple (i, j) , $i = 1, \dots, I$ and $j = 1, \dots, J$, we observe a quantitative variable Y_{ij} , n_{ij} times. We assume that Y_{ij} follows a normal distribution $\mathcal{N}(\mu_{ij}, \sigma^2)$ in each subpopulation defined by values i and j of the two factors.
- **Model:** The model is written

$$Y_{ijk} = \mu_{ij} + \epsilon_{ijk}, \quad \text{for } k = 1, \dots, n_{ij}, \quad i = 1, \dots, I, \quad j = 1, \dots, J,$$

where the errors ϵ_{ijk} are independent random variables from a $\mathcal{N}(0, \sigma^2)$ distribution.

In this model, the real parameters $\mu_{11}, \dots, \mu_{I1}, \dots, \mu_{1J}, \dots, \mu_{IJ}$ are unknown, as is the variance σ^2 .

We decompose μ_{ij} so that the effects of the factors A and B appear, as well as the effect of their interaction:

$$\mu_{ij} = \mu_{\bullet\bullet} + \alpha_i^A + \alpha_j^B + \beta_{ij}$$

with:

- $\mu_{\bullet\bullet} = \frac{1}{IJ} \sum_{i=1}^I \sum_{j=1}^J \mu_{ij}$: general mean effect
- $\mu_{i\bullet} = \frac{1}{J} \sum_{j=1}^J \mu_{ij}$: effect of level i of factor A
- $\alpha_i^A = \mu_{i\bullet} - \mu_{\bullet\bullet}$: differential effect of level i of factor A
- $\mu_{\bullet j} = \frac{1}{I} \sum_{i=1}^I \mu_{ij}$: effect of level j of factor B
- $\alpha_j^B = \mu_{\bullet j} - \mu_{\bullet\bullet}$: differential effect of level j of factor B
- $\beta_{ij} = \mu_{ij} - \mu_{\bullet\bullet} - \alpha_i^A - \alpha_j^B$: interaction effect of level i of factor A and level j of factor B

Note

By construction, $\sum_{i=1}^I \alpha_i^A = 0$, $\sum_{j=1}^J \alpha_j^B = 0$ and $\forall i$, $\sum_{j=1}^J \beta_{ij} = 0$, $\forall j$, $\sum_{i=1}^I \beta_{ij} = 0$.

The aim here is to detect:

- whether factor A has an effect on the quantitative variable Y
- whether factor B has an effect on the quantitative variable Y
- and whether there is an interaction effect between factors A and B on the quantitative variable Y

15.2.2 Example and Graphical Inspection

- *Example of use:* The next table gives the yield of einkorn wheat from fields in four different regions with three types of fertilizer.

	Region I	Region II	Region III	Region IV
Fertilizer E_1	15 14 17	21 20 21	14 15 14	16 17 17
Fertilizer E_2	16 19 20	23 24 25	15 14 14	12 11 12
Fertilizer E_3	18 17 17	20 21 21	17 19 17	12 13 13

These data are input in R with the following instructions:

```
> yield <- c(15,14,17,21,20,21,14,15,14,16,17,17,16,19,20,23,
+           24,25,15,14,14,12,11,12,18,17,17,20,21,21,17,19,
+           17,12,13,13)
> fertilizer <- gl(3,12,36,labels=paste("Fertilizer",1:3))
> region <- gl(4,3,36,labels=paste("Region",1:4))
> wheat <- data.frame(yield,fertilizer,region)
```

We wish to study the effect of the type of fertilizer (E_1 , E_2 and E_3) on the yield per hectare of einkorn wheat and to find out whether there is a significantly different yield between the four regions.

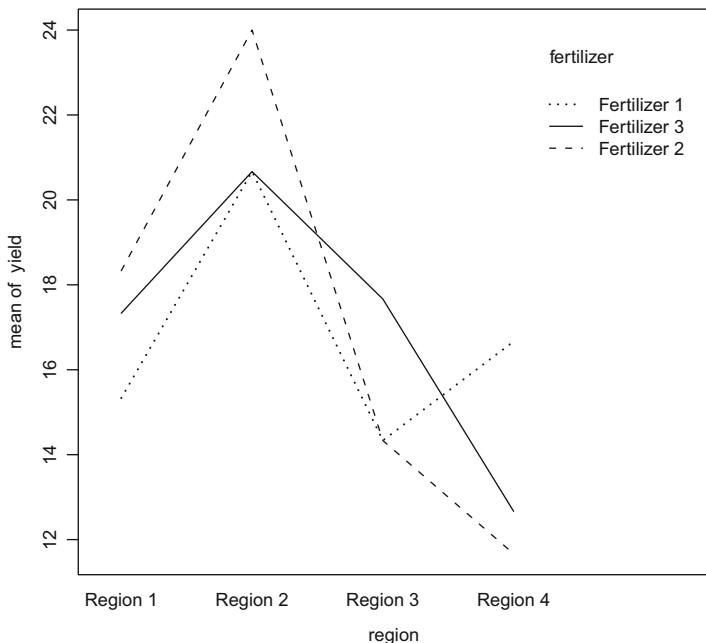
Note

In this example, the data are balanced (there is the same number of observations for each pair of factor levels). When this is not the case, the ANOVA table is no longer unique; we advise you to use a decomposition of variance of type III (see [31]).



- **Graphical inspection:** In the ANOVA model with two factors and interaction, the effect of a factor on the explained variable can be different depending on the modalities of the other factor. This flexibility in the model can be visualized. The command to explore this interaction is `interaction.plot()`. The function `plotMeans()`, available in the package Rcmdr, also allows graphical inspection of the interaction (Fig. 15.3).

```
> interaction.plot(region,fertilizer,yield)
```



```
> interaction.plot(fertilizer,region,yield)
```

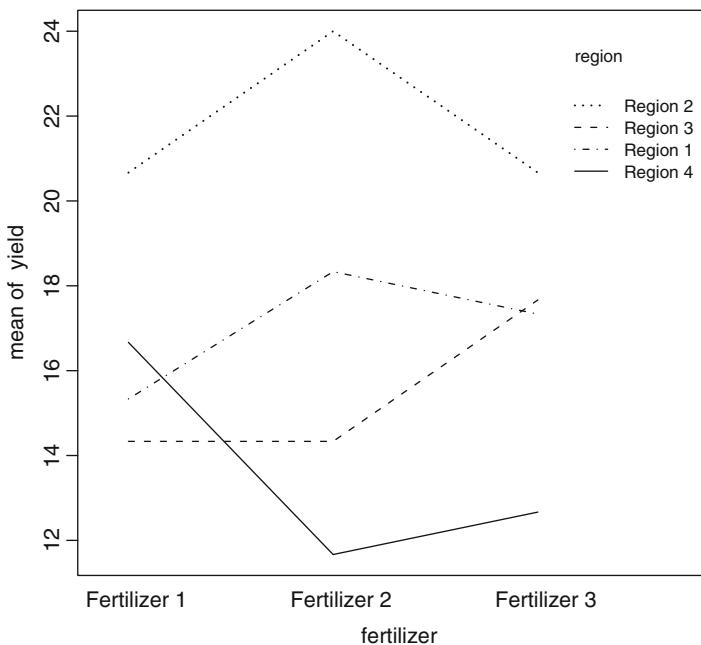


Fig. 15.3: Exploration of interaction in two-way ANOVA

These plots indicate that there seems to be interaction (the lines intersect).

15.2.3 ANOVA Table, Tests and Parameter Estimation

- **R instruction for ANOVA table:** Several functions exist to perform two-way ANOVA with interaction: `aov()`, `anova(lm())` and `Anova()` (in package `car`).

Warning

When you have only one observation per combination of modalities of the factors A and B (i.e. $n_{ij} = 1 \forall i, j$), you can only estimate two-way ANOVA without interaction.

`aov(yield~region+fertilizer)`.

```
> model2 <- summary(aov(yield~region*fertilizer,data=wheat))
> model2
      Df Sum Sq Mean Sq  F value    Pr(>F)
region          3 327.19 109.065 112.1810 2.955e-14 ***
fertilizer       2   0.89   0.444   0.4571   0.6385
region:fertilizer 6  99.56  16.593  17.0667 1.359e-07 ***
Residuals      24  23.33   0.972
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
> anova(lm(yield~region*fertilizer,data=wheat))
Analysis of Variance Table
Response: yield
      Df Sum Sq Mean Sq F value    Pr(>F)
region        3 327.19 109.065 112.1810 2.955e-14 ***
fertilizer     2   0.89   0.444   0.4571   0.6385
region:fertilizer 6  99.56  16.593  17.0667 1.359e-07 ***
Residuals    24  23.33   0.972
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> Anova(lm(yield~region*fertilizer,data=wheat))
Anova Table (Type II tests)
Response: yield
      Sum Sq Df F value    Pr(>F)
region       327.19  3 112.1810 2.955e-14 ***
fertilizer     0.89  2   0.4571   0.6385
region:fertilizer 99.56  6  17.0667 1.359e-07 ***
Residuals    23.33 24
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note

The formula `region*fertilizer`, used in `aov()` and `lm()`, corresponds in fact to the formula `region+fertilizer+region:fertilizer`, i.e. the factor region, the factor fertilizer and the interaction of these two factors.



The *p*-value associated with the test of interaction is significant. This implies that the effect of fertilizer of yield can be different depending on the region.

Warning

We consider that there is no interaction effect if the associated *p*-value is greater than 5 %. In this case, we perform an ANOVA without an interaction term which makes it easier to interpret the principal effect. When there is interaction, do not interpret the principal effects in the ANOVA table output.



- **Test of conditional effects with interaction:** For example, we wish to know whether there is a fertilizer effect in region 1. To this end, we use the function `subset()`, which only uses data from a given region.

```
> fertilizer.region1 <- summary(aov(yield~fertilizer,subset=
+                                     region=="Region 1"))
> fertilizer.region1
      Df Sum Sq Mean Sq F value    Pr(>F)
fertilizer     2      14   7.0000      3   0.125
Residuals     6      14   2.3333
```

Warning

The test in this ANOVA table corresponds to ANOVA with one factor (fertilizer) of the yield of wheat in region 1. It does not take into account any information from data in the other regions, which would allow for a better estimation of the residual variance. To test the fertilizer effect in region 1, divide the mean square of the fertilizer factor found in the ANOVA restricted to region 1 by the mean residual square of the ANOVA with interaction:

```
> F.fertilizer.region1 <- fertilizer.region1[[1]]$Mean[1] /  
+                               model2[[1]]$Mean[4]  
> pvalue <- 1-pf(F.fertilizer.region1,df1=2,df2=24)  
> pvalue  
[1] 0.003552714
```

The p -value is less than 5 %; hence we conclude that there is a fertilizer effect in region 1.

- *Parameter estimation:* The parameters can be estimated with the function `summary()` for the model `lm(yield~region*fertilizer)`. Recall that R imposes the constraints $\alpha_1^A = 0$, $\alpha_1^B = 0$, $\beta_{1j} = 0 \forall j = 1, \dots, J$ and $\beta_{i1} = 0 \forall i = 1, \dots, I$.

```
> summary(lm(yield~region*fertilizer))  
Call:  
lm(formula = yield ~ region * fertilizer)  
Residuals:  
    Min      1Q  Median      3Q      Max  
-2.3333 -0.6667  0.1667  0.3333  1.6667  
Coefficients:  
                                         Estimate Std. Error  
(Intercept)                         15.3333   0.5693  
regionRegion 2                          5.3333   0.8051  
regionRegion 3                         -1.0000   0.8051  
regionRegion 4                          1.3333   0.8051  
fertilizerFertilizer 2                  3.0000   0.8051  
fertilizerFertilizer 3                  2.0000   0.8051  
regionRegion 2:fertilizerFertilizer 2   0.3333   1.1386  
regionRegion 3:fertilizerFertilizer 2   -3.0000   1.1386  
regionRegion 4:fertilizerFertilizer 2   -8.0000   1.1386  
regionRegion 2:fertilizerFertilizer 3   -2.0000   1.1386  
regionRegion 3:fertilizerFertilizer 3   1.3333   1.1386  
regionRegion 4:fertilizerFertilizer 3   -6.0000   1.1386  
                                         t value Pr(>|t|)  
(Intercept)                         26.935 < 2e-16 ***  
regionRegion 2                        6.625  0.000000749 ***  
regionRegion 3                        -1.242   0.22619  
regionRegion 4                        1.656   0.11071  
fertilizerFertilizer 2                 3.726   0.00105 **  
fertilizerFertilizer 3                 2.484   0.02036 *  
regionRegion 2:fertilizerFertilizer 2   0.293   0.77221  
regionRegion 3:fertilizerFertilizer 2   -2.635   0.01451 *
```

```

regionRegion 4:fertilizerFertilizer 2  -7.026  0.000000290 ***
regionRegion 2:fertilizerFertilizer 3  -1.757      0.09174 .
regionRegion 3:fertilizerFertilizer 3   1.171      0.25306
regionRegion 4:fertilizerFertilizer 3  -5.270  0.000021016 ***
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.986 on 24 degrees of freedom
Multiple R-squared:  0.9483,    Adjusted R-squared:  0.9245
F-statistic: 39.99 on 11 and 24 DF,  p-value: 1.009e-12

```

Thus the intercept corresponds to the estimate of the mean yield for fertilizer 1 in region 1. For example, the coefficient associated with region 2 (6.625) corresponds to the estimate of the difference of the mean yield with fertilizer 1 of region 2 and the mean yield with fertilizer 1 of region 1. The Student tests associated with the factors can therefore be interpreted. However, those associated with the estimates of the crossed factors coefficients are not relevant.

Note

In this example, there is no particular reason of choosing this constraint. To change it, use the function `C()`. For example,

`summary(lm(yield~C(region,sum)*C(fertilizer,sum)))`
 corresponds to the constraints $\sum_{i=1}^I \alpha_i^A = 0$, $\sum_{j=1}^J \alpha_j^B = 0$ and
 $\forall i, \sum_{j=1}^J \beta_{ij} = 0, \forall j, \sum_{i=1}^I \beta_{ij} = 0$.



15.2.4 Validating Assumptions

- *Validating assumptions:* As in one-way ANOVA, we validate the model with a study of the residuals of the underlying linear model (Fig. 15.4).

```

> par(mfrow=c(2,2))
> plot(model,col.smooth="red")

```

However, if the data size is large enough for each pair of factor modalities, it is better to check for normality in each subpopulation and for homoscedasticity.

15.2.5 Contrasts

- *Contrast method:* We refer the reader to the definition of contrasts given for one-way ANOVA.

For example, suppose we wish to know whether there is a significant difference of yield in region 1 between fertilizers 1 and 2. We perform a contrast test using the function `estimable()` from package `gmodels`.

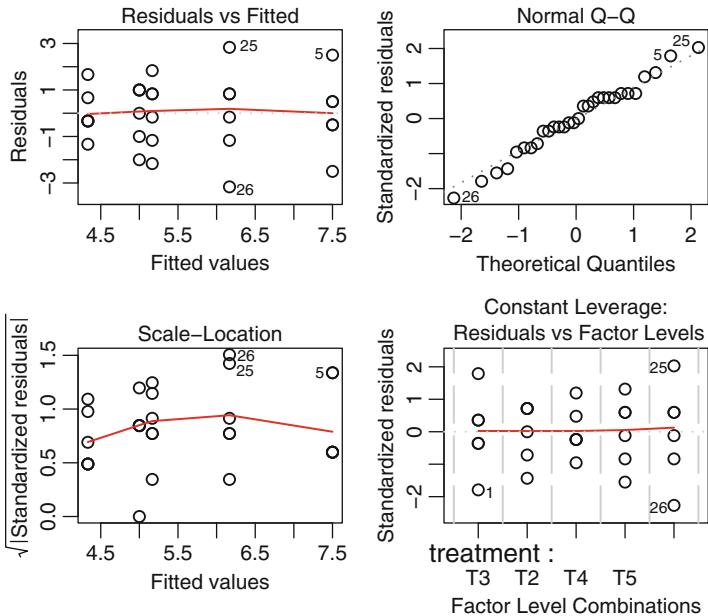


Fig. 15.4: Residual analysis in two-way ANOVA

```
> mod.inter <- lm(yield ~ fertilizer:region-1)
> cm <- rbind("F1 vs F2 in R1"=
+               c(1,-1,0,0,0,0,0,0,0,0,0,0))
> estimable(mod.inter,cm)
      Estimate Std. Error   t value DF Pr(>|t|/)
F1 vs F2 in R1      -3  0.8050765 -3.726354 24 0.001048837
```

We get the same *p*-value as with the test for *estimating model parameters*. Another example of use of the method of contrasts is the comparison of the yield with fertilizer 1 in the southern regions (regions 1 and 2) with the northern regions (regions 3 and 4):

```
> cm <- rbind("F1 vs F2 in R1"=
+               c(1,-1,0,0,0,0,0,0,0,0,0),
+               "R1 & R2 vs R3 & R4 for F1" =
+               c(1,0,0,1,0,0,-1,0,0,-1,0,0))
> estimable(mod.inter,cm)
      Estimate Std. Error   t value DF Pr(>|t|/)
F1 vs F2 in R1      -3  0.8050765 -3.726354 24 0.0010488374
R1 & R2 vs R3 & R4 for F1      5  1.1385501  4.391550 24
                                         Pr(>|t|/)
F1 vs F2 in R1      0.0010488374
R1 & R2 vs R3 & R4 for F1 0.0001951599
```

15.2.6 Summary

This table lists the main functions for two-way ANOVA (Table 15.2).

Table 15.2: Main functions for two-way ANOVA

R instruction	Description
<code>interaction.plot(Y, factor(X), factor(Z))</code>	Graphical inspection
<code>aov(Y~factor(X)*factor(Z))</code>	Two-way ANOVA with interaction
<code>summary(aov(Y~factor(X)*factor(Z)))</code>	ANOVA table
<code>anova(lm(Y~factor(X)*factor(Z)))</code>	ANOVA table
<code>Anova(lm(Y~factor(X)*factor(Z)))</code>	ANOVA table (package car)

Warning

For two-way ANOVA with unbalanced data, you should use a decomposition of the sums of squares of type III (see [31]).

```
> model.lm <- lm(yield~region*fertilizer,contrasts=list(region=
+                      contr.sum,fertilizer=contr.sum))
> Anova(model.lm,type="III")
Anova Table (Type III tests)
Response: yield
      Sum Sq Df   F value    Pr(>F)
(Intercept) 10370.0  1 10666.3143 < 2.2e-16 ***
region        327.2  3   112.1810 2.955e-14 ***
fertilizer     0.9  2     0.4571   0.6385
region:fertilizer  99.6  6    17.0667 1.359e-07 ***
Residuals     23.3 24
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



Remember to use the option `contrasts` in function `lm()`.

SECTION 15.3

Repeated Measures Analysis of Variance

This section is a brief introduction to repeated measures ANOVA models. We first introduce some vocabulary to better understand the three models introduced in this section.

A fixed effects model is a model where all explanatory variables (factors) are treated as non-random (e.g., controlled). A random effects model or mixed effects model occurs when some or all explanatory variables are assumed to be random.

In ANOVA, a statistical unit is called a “subject”. When a dependent variable is measured on groups of independent subjects, where each group is exposed to a different condition, the set of conditions is called a between-subjects factor. The models in Sects. 15.1 and 15.2 only use such factors.

In the case where each subject has a measure of the dependent variable for all modalities of the factor, this factor is called a within-subjects factor. An experiment has within-subjects design when at least one factor is within-subjects. Such experimental designs are also called repeated measures designs, since within-subjects factors always imply repeated measures on each subject. By construction, this is equivalent to considering an extra random factor, the subject factor.

When an analysis includes both within-subjects and between-subjects factors, it is called a repeated measures ANOVA with between-subjects factors, mixed-design ANOVA or split-plot ANOVA.

15.3.1 One-Way Repeated Measures ANOVA

- **Aim:** We consider the case where for each subject s (out of n), we measure the response (dependent) variable Y for each of the I modalities of the fixed-effects factor X .
- **Example:** For 15 minutes, we count how many times each of seven rats presses a lever, in three reinforcement conditions. In the first condition, the rat is given well-liked food, in the second average-liked food and in the third less-liked food. The results are given in the following table:

Subject	Factor: condition		
	cond ₁	cond ₂	cond ₃
s_1	8	6	2
s_2	6	5	1
s_3	7	5	0
s_4	9	3	3
s_5	5	4	1
s_6	7	5	2
s_7	6	2	0

- **Model:** The underlying model is a mixed-effects model:

$$Y_{si} = \mu_i + \pi_s + \epsilon_{si}, \quad s = 1, \dots, n, i = 1, \dots, I$$

where μ_i measures the fixed effect of modality i of factor X , the π_s are independent random variables following a $\mathcal{N}(0, \sigma_\pi^2)$ distribution, to take into account the dependence between measures made on individual s , and the ϵ_{si} are independent

random variables following a $\mathcal{N}(0, \sigma^2)$ distribution. We also assume that the π_s and the ϵ_{si} are independent. By construction, this single-factor model with repeated measures is in fact a two-factor model: one fixed within-subjects factor (X) and one random factor, the subject factor.

► **R instructions:**

```
summary(aov(Y ~ X + Error=subject/X,data=my.data.frame))
```

where `my.data.frame` is a `data.frame` containing variables `Y` and `X`, and a variable `subject` giving the subject ID number. The variables `X` and `subject` must necessarily be defined as factors.

Note

You can also use the function `lme()` included in package `nlme()`:

```
anova(lme(Y ~ X,random= 1|subject,data=my.data.frame))
```



► *Back to the example:*

```
> rat <- data.frame(lever=c(8,6,2,6,5,1,7,5,0,9,3,3,5,4,1,
+                      7,5,2,6,2,0),subject=g1(7,3,21),cond=g1(3,1,21))
> summary(aov(lever~cond+Error(subject/cond), data=rat))
Error: subject
      Df Sum Sq Mean Sq F value Pr(>F)
Residuals 6 15.905 2.6508
Error: subject:cond
      Df Sum Sq Mean Sq F value     Pr(>F)
cond      2 108.86 54.429 47.297 0.000002036 ***
Residuals 12 13.81  1.151
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

15.3.2 Two-Factor Model with Repeated Measures for Both Factors

- **Aim:** We consider the case where for each subject s (out of n), the response variable Y is measured for each of the $I \times J$ combinations of modalities of two fixed effects factors A and B .
- **Example:** A scientist wishes to study the effect of lecithin on memory issues. Four subjects receive a daily treatment. After one, two and six months of treatment, each subject undergoes two tests (Test 1 and Test 2). The results are shown in the following table:

Subject	Test 1			Test 2		
	M ₁	M ₂	M ₆	M ₁	M ₂	M ₆
s ₁	10	11	9	3	6	3
s ₂	18	20	17	16	20	14
s ₃	6	8	8	5	6	3
s ₄	4	9	9	10	10	6

► *Model:* The underlying model is a mixed model:

$$Y_{sij} = \mu_{\bullet\bullet} + \alpha_i^A + \alpha_j^B + \beta_{ij} + \pi_s + \pi_{si}^A + \pi_{sj}^B + \epsilon_{sij}$$

with $s = 1, \dots, n$, $i = 1, \dots, I$, $j = 1, \dots, J$ and where the terms $\mu_{\bullet\bullet}$, α_i^A , α_j^B and β_{ij} were defined in Sect. 15.2. The random effect subject is represented by the i.i.d. random variables π_s with distribution $\mathcal{N}(0, \sigma_\pi^2)$. The i.i.d. random variables $\pi_{si}^A \sim \mathcal{N}(0, \sigma_{\pi A}^2)$ measure the random interaction effects between the subject factor and the fixed factor A. The i.i.d. random variables $\pi_{sj}^B \sim \mathcal{N}(0, \sigma_{\pi B}^2)$ measure the random interaction effects between the subject factor and the fixed factor B. The errors ϵ_{sij} are i.i.d. variables following the distribution $\mathcal{N}(0, \sigma^2)$. Furthermore, we assume that the errors are independent of the π_s , π_{si}^A and π_{sj}^B .

► *R instructions:*

```
summary(aov(Y ~ A*B + Error(subject/(A*B)), data=my.
data.frame))
```

► *Back to the example:*

```
> lecithin <- data.frame(memory=c(10,11,9,3,6,3,18,20,17,16,20,
+                               14,6,8,8,5,6,3,4,9,9,10,10,6), subject=gl(4,6,24),
+                               test=gl(2,3,24), month=gl(3,1,24))
> summary(aov(memory~month*test+Error(subject/(test*month)),
+             data=lecithin))
Error: subject
      Df Sum Sq Mean Sq F value Pr(>F)
Residuals 3 508.13 169.38
Error: subject:test
      Df Sum Sq Mean Sq F value Pr(>F)
test       1 30.375 30.375 2.2158 0.2333
Residuals 3 41.125 13.708
Error: subject:month
      Df Sum Sq Mean Sq F value    Pr(>F)
month      2 32.25 16.1250 16.826 0.003465 ***
Residuals 6   5.75  0.9583
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Error: subject:test:month
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
month:test	2	12.25	6.125	2.3333	0.1780
Residuals	6	15.75	2.625		

15.3.3 Two-Factor Model with Repeated Measures for One Factor

- *Aim:* We are interested in the case where subjects are allocated to groups defined by the I modalities of fixed factor A. For each subject, we measure the response variable for all J modalities of fixed factor B.
- *Example:* In an experiment, the subjects estimate the length of a metal bar. The bars are of three different lengths. Two groups of four distinct subjects are created. In each group, each subject is shown three bars of different lengths.

Subject	G1		
	L1	L2	L3
1	10	11	9
2	18	20	17
3	6	8	8
4	4	9	9

Subject	G2		
	L1	L2	L3
1	3	6	3
2	16	20	14
3	5	6	3
4	10	10	6

- *Model:*

$$Y_{s(i)j} = \mu_{\bullet\bullet} + \alpha_i^A + \alpha_j^B + \beta_{ij} + \pi_{s(i)}^A + \epsilon_{s(i)j}, \quad s = 1, \dots, n, i = 1, \dots, I, j = 1, \dots, J$$

where the $\pi_{s(i)}$ are i.i.d. random variables with distribution $\mathcal{N}(0, \sigma_\pi^2)$ measuring the random effects of the modalities s of the subject factor. The errors $\epsilon_{s(i)j}$ are i.i.d. random variables with distribution $\mathcal{N}(0, \sigma^2)$. Furthermore, we assume that the $\epsilon_{s(i)j}$ are independent of the $\pi_{s(i)}$. The terms $\mu_{\bullet\bullet}$, α_i^A , α_j^B , β_{ij} were defined in Sect. 15.2. For the model to be identifiable, we have to impose the constraints $\sum_{i=1}^I \alpha_i^A = 0$, $\sum_{j=1}^J \alpha_j^B = 0$, $\forall j, \sum_{i=1}^I \beta_{ij} = 0$, $\forall i, \sum_{j=1}^J \beta_{ij} = 0$. The realization $y_{s(i)j}$ of the random variable $Y_{s(i)j}$ represents the observation of the s th subject from i th group of factor A, for level j of factor B. The notation $s(i)$ underlines the fact that the subject factor is nested in factor A.

- *R instructions:*

```
summary(aov(Y~A*B + Error(subject %in% A), data=my.data.frame))
or equivalently:
summary(aov(Y ~ A*B + Error(subject:A), data=my.data.frame))
```

► Back to the example:

```
> bar.estim <- data.frame(bar=c(10,11,9,3,6,3,18,20,17,16,20,
+                                14,6,8,8,5,6,3,4,9,9,10,10,6),subject=gl(4,6,24),
+                                group=gl(2,3,24), # Factor A.
+                                long=gl(3,1,24)   # Factor B.
+ )
> summary(aov(bar ~ group*long +
+               Error(subject %in% group),data=bar.estim))
Error: subject:group
      Df Sum Sq Mean Sq F value Pr(>F)
group       1 30.37  30.375  0.3318 0.5855
Residuals   6 549.25  91.542
Error: Within
      Df Sum Sq Mean Sq F value    Pr(>F)
long        2 32.25 16.1250  9.0000 0.004096 ***
group:long  2 12.25  6.1250  3.4186 0.066833 .
Residuals 12 21.50  1.7917
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Memorandum

`aov()`: perform ANOVA
`anova(lm())`, `Anova(lm())`: ANOVA table
`factor()`, `as.factor()`: declare a variable as a factor
`C()`: specify the constraint in ANOVA
`bartlett.test()`, `levene.test()`: tests of equality of variances
`pairwise.t.test()`: pairwise comparisons
`fit.contrast()`: contrast test (package `gregmisc`)
`estimable()`: contrast test (package `gmodels`)
`interaction.plot()`: graphical inspection for interaction
`Error()`: create a formula to specify the nesting of the subject factor



Exercises

- 15.1-** Give the instruction to perform ANOVA with one factor (noted A).
- 15.2-** Give the instruction to perform ANOVA with two factors (noted A and B) with interaction.
- 15.3-** Which test would you use to check for homoscedasticity in an ANOVA model?
- 15.4-** Which instruction performs pairwise tests after single-factor ANOVA?
- 15.5-** Which function gives the estimates for a single-factor ANOVA model?
- 15.6-** Which function is used to choose the constraint in ANOVA?



Worksheet

A- Study of One-Way ANOVA

- *Study of noise levels:* In order to study the influence of the factor “level of surrounding noise” on the ability of a subject to solve a problem, the following experiment is designed: 24 school children are randomly allocated to four rooms. Street noises were previously recorded and are played in each room at a specific noise level. The children must solve a series of problems. The response variable is the final grade for the series of problems. The results are given in the following table:

Noise level			
1	2	3	4
62	56	63	68
60	62	67	66
63	60	71	71
59	61	64	67
63	63	65	68
59	64	66	68

We wish to know whether there is an effect of the factor “level of surrounding noise” on a subject’s ability to solve a problem.

- 15.1- Input the data set in an adequate structure to perform ANOVA.
- 15.2- Write down the ANOVA model to answer the question.
- 15.3- Perform the analysis corresponding to your model.
- 15.4- Perform all pairwise comparisons of noise levels; remember to take into account the problem of multiplicity of tests.

• *Study of intima–media:* In the “Intima–media” study, we are interested in the relationship between intima–media thickness and alcohol consumption.

- 15.1- Download the intima-media data file.
- 15.2- Suggest a plot to visualize the differences in intima–media thickness depending on alcohol consumption.
- 15.3- Is there a difference of mean intima–media thickness depending on alcohol consumption?
- 15.4- Analyse the residuals to validate the assumptions of your statistical study.

• *Study of physical activity:* In a study of risky behaviour amongst people aged 14 to 25, a scientist observed offences (theft, racketeering, brawls, ...) committed by youngsters and weekly physical activity. Risky behaviour was measured on a scale from 0 to 100. A subset of the data is available at the URL <http://www.biostatisticien.eu/springeR/sports.RData>:

```
> print(sports[sample(1:105,10),],row.names=FALSE)
  score   time
    9 [2;3[
   75 [0;1[
   0 [3;4[
   21 [2;3[
   67 [4;5[
   69 [1;2[
   36 [2;3[
   0 [3;4[
   87 [0;1[
   16 [2;3[
```

- 15.1- Describe the factors involved and write down the model (and the underlying assumptions).

- 15.2-** Perform a test at the 5 % level to decide whether there is a significant effect between physical activity and risky behaviour.

We call “not athletic” youngsters who play sports for less than 2 hours per week, “somewhat athletic” those with weekly physical activity in the interval [2,4) and “very athletic” those who play sports at least 4 hours per week.

The scientist makes the following research assumptions:

- Research assumption 1: “Not athletic” youngsters have more risky behaviour than “somewhat athletic” youngsters.
- Research assumption 2: Risky behaviour is different between “very athletic” and “not athletic” youngsters.

- 15.3-** Translate the research assumptions into contrasts.

- 15.4-** Test these two assumptions at the 5 % level.

B- Study of Two-Way ANOVA

- *Study of batteries:* In an experiment on battery lifetime, the aim is to find battery life as a function of type of battery. It is known that battery lifetime depends on temperature, so a plane with two factors (temperature and type of battery) is created. The following table gives battery lifetime depending on these two factors.

	15 °C	70 °C	125 °C
Type I	130 155	34 40	20 70
	74 180	80 75	82 58
Type II	150 188	136 122	25 70
	159 126	106 115	58 45
Type III	138 110	174 120	96 104
	168 160	150 139	82 60

- 15.1-** Which factors are involved in this experiment? What are their modalities? What is the response variable?

- 15.2-** Propose and define an ANOVA model for this data set.

- 15.3-** Propose a graphical representation to visualize any interaction there may be in the model.

- 15.4-** Estimate the parameters of the model.

- 15.5-** Draw an ANOVA table for your model.

- 15.6-** Perform the relevant tests to finalize this analysis.

- *Milk yield:* We are interested in the influence of type and quantity of food on milk yield. We have observed these forty values:

	Straw	Hay	Grass	Silage
Low dose	8 11 11 10 7	12 13 14 11 10	10 12 12 13 14	17 13 17 14 13
High dose	8 9 8 10 9	10 7 10 12 11	11 9 11 11 12	13 12 11 15 14

- 15.1-** Propose and define an ANOVA model to study the influence of type and quantity of food on milk yield.

- 15.2-** Propose a graphical representation to visualize any interaction there may be in the model.
- 15.3-** Estimate the parameters of the model.
- 15.4-** Draw an ANOVA table for your model.
- 15.5-** Perform the relevant tests to finalize this analysis.

• *Intima-media study:* In practical A, we looked at a possible relationship between intima–media thickness and alcohol consumption in the “Intima–media” study. We now wonder whether intima–media thickness is linked to alcohol consumption and tobacco consumption.

- 15.1-** Download the intima–media data set.
- 15.2-** Propose and define an ANOVA model to study the influence of tobacco consumption and alcohol consumption on intima–media thickness.
- 15.3-** Propose a graphical representation to visualize any interaction there may be in the model.
- 15.4-** Is there a difference in intima–media thickness depending on alcohol consumption? Depending on tobacco consumption?

Appendix: Installing R and R Packages

Prerequisites and goals of this chapter

- No prerequisite. You may be interested in reading Chap. 3 first.
- This chapter explains how to install R version x (replace throughout x by the number of the latest version) under Microsoft Windows and how to install additional packages under Windows or Linux.

— SECTION A.1 —

Installing R Under Microsoft Windows

First download R (file R- x -win.exe where x is the number of the latest version) using your usual web browser from the URL <http://cran.r-project.org/bin/windows/base/>. Save this executable file on the Windows Desktop and double-

click the file R- x -win.exe (its icon is ).

The software then installs. All you have to do is follow the instructions displayed on your screen and keep the default options.

When the icon  is added to the Desktop, installation is complete.

SECTION A.2 —

Installing Additional Packages

Many additional modules (called packages or libraries) are available on the website http://cran.r-project.org/web/packages/available_packages_by_name.html or here: <http://cran.r-project.org/bin/windows/contrib/>, in the folder corresponding to the number x of your R version.

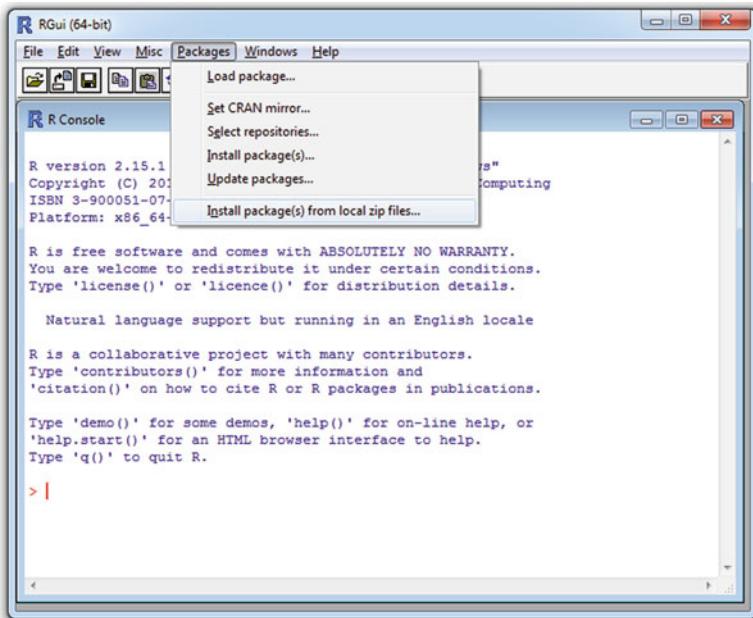
Packages extend the functionalities of R. We present several ways of installing a new package.

A.2.1 *Installing from a File on Your Disk*

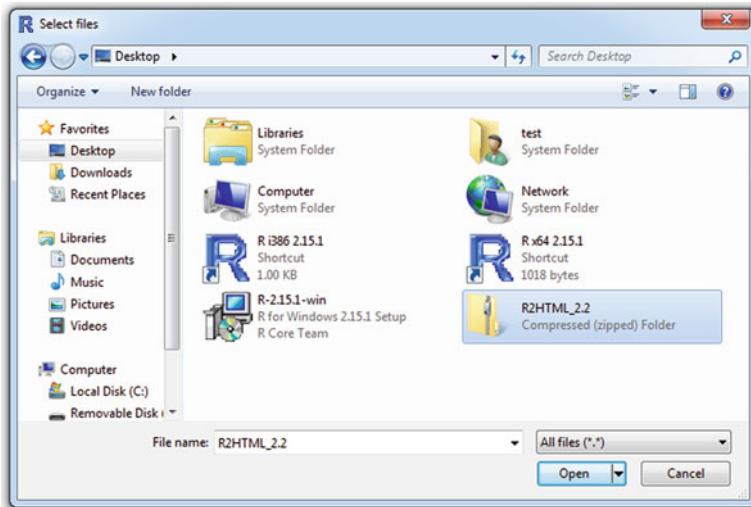
For example, you can download from the website mentioned above the file: R2HTML_number.zip and save it on the Windows Desktop.

To install this package, first start R by double-clicking its icon .

Go to the menu Packages, in the submenu Install package(s) from zip files...



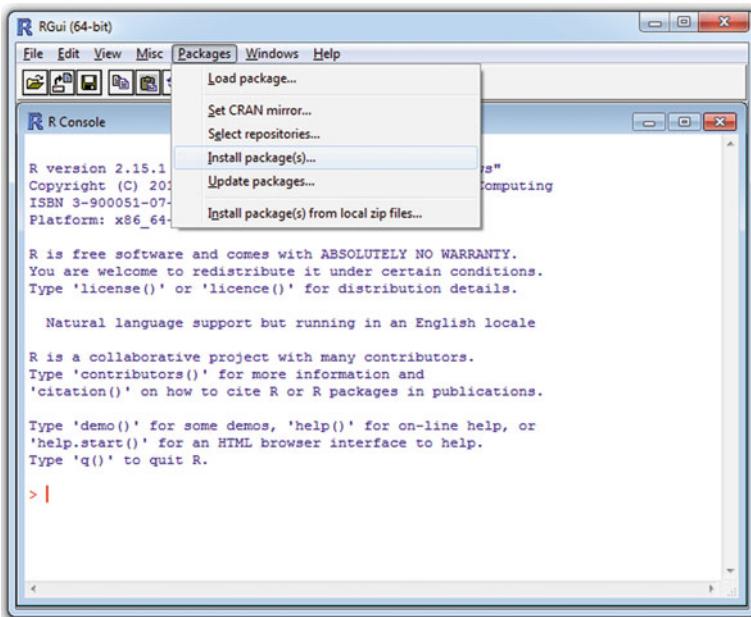
Select the file R2HTML_number.zip on the Windows Desktop, then click on “Open”.



A.2.2 Installing Directly from the Internet

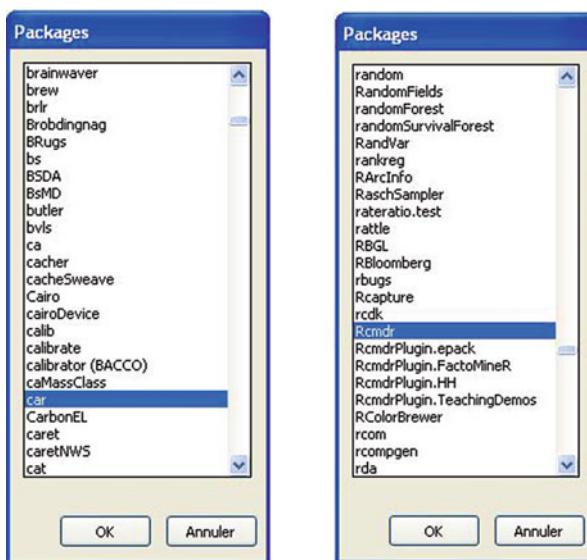
For example, to install packages `car` and `Rcmdr`, first launch **R** by double-clicking its icon on the Desktop.

Then go to menu **Packages** and submenu **Install package(s)...**

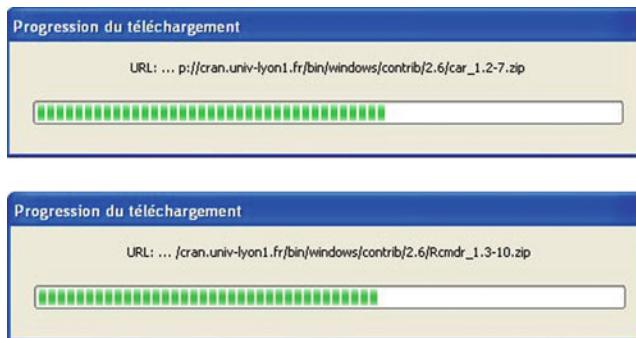


Choose a CRAN mirror close to your location and click OK.

At the next step, select the entries “car” and “Rcmdr”. To do this, first click on “car”, then scroll down and click on “Rcmdr” while pressing the CTRL key. Check that both entries have been selected (highlighted in blue).



Click “OK”. You should see the two following screens, indicating that the packages you selected are being installed.



Warning

This procedure may fail, for example, if your system administrator has blocked access to some websites with a firewall, has imposed a proxy to surf the Internet or has forbidden writing in some Windows folders. If you encounter a problem, we advise you to contact your administrator. Note that you can force R to use a proxy and that you can install packages locally in your own home directory. You can consult sections 2.15 (*How do I set environment variables?*) or 2.19 (*The Internet download functions fail*) of the Windows FAQ available here: <http://cran.r-project.org/bin/windows/base/rw-FAQ.html>



A.2.3 *Installing from the Command Line*

You can use R without the menus of the graphical user interface. This is the case for example under Unix/Linux, where R does not have a graphical user interface. In that case, use the following commands to install packages:

- To install a package from a *.zip file on your hard disk:

```
install.packages(choose.files(), repos = NULL)
```

- To install a package (say Rcmdr) from the CRAN website:

```
install.packages("Rcmdr")
```

Another possibility is to install a package without going through R, directly from an MS-DOS command window under Microsoft or from a terminal under Linux or MacOS. In that case, you will need many compilation tools. If these tools are not installed on your computer, refer to the section on package creation in Chap. 9.

If these tools are installed on your computer, you can try this:

For example, download the file (package source) Rcmdr_number.tar.gz from the URL <http://cran.r-project.org/web/packages/Rcmdr>. Save it (on the Desktop) and start an MS-DOS window (Start menu/Run/cmd), then enter:

```
cd Desktop
```

```
R CMD INSTALL Rcmdr_number.tar.gz (replace number with the relevant number).
```

A.2.4 *Installing Packages Under Linux*

Note that the commands of the previous section also work under Linux. But sometimes, you need to be logged in as root to use them (command su - in a terminal window).

If you do not have root access, you can still install packages locally, in your home directory. For example, for package Rcmdr, type in a terminal:

```
R CMD INSTALL --library=/home/user/Rlibs Rcmdr_number.tar.gz
```

(you should first have created the folder Rlibs with the command mkdir Rlibs, and should replace /home/user/Rlibs with the appropriate path and number with the appropriate number).

Finally, so that R knows where to look for installed packages, you must create a file `~/.Renviron` containing the line

```
R_LIBS=/home/user/Rlibs
```

Tip

If your computer is behind a firewall and you need to use a proxy to go online, you can use the following command to install a package directly from R:



```
Sys.setenv("http_proxy"="http://user:pass@url_of_
the_proxy:num_port")
install.packages("Rcmdr",method="wget")
```

For further details, read the online help for function `download.file()`.

— SECTION A.3 —

Loading Installed Packages

Warning

To understand this section, you need to have a rough understanding of the difference between your computer's random access memory (RAM) and physical memory.

Installing a package means that its files are “written” physically on the hard disk: when you turn your computer off then on again, the files are still in the same place. You will not need to reinstall the package, unless you need an updated version. On the contrary, loading a package (to the memory) means that it is temporarily at the user's disposal in R. But if you close and reopen R, the package is no longer available: you need to load it again.

To sum up, once you have installed a package on your computer's hard disk, you have to load it to R's memory before you can use it.

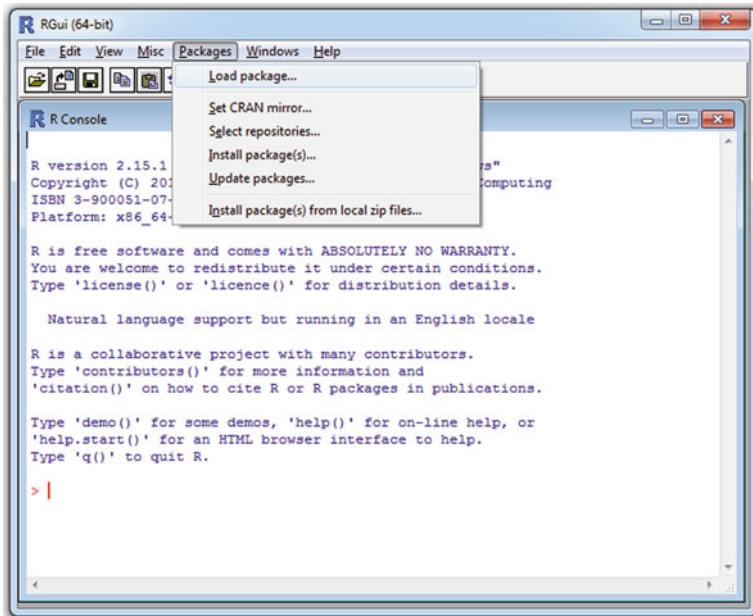
For example, if you type in the R console:

`Commander()`

you should see the following error message, which indicates that the package including this function cannot be accessed by R:

`Error: cannot find function "Commander"`

You must first load Rcmdr to the memory. To do this, you can either type `require("Rcmdr")` in the console or go to the menu Packages/Load package...,



and use the mouse to load package Rcmdr.



The following window then appears.



Close it, and type once again in the R console:

```
Commander()
```

Note that the package `Rcmdr` has been loaded and that this command no longer returns an error message.

Tip

Note that R offers the possibility of automatically loading some packages upon start-up:

```
.First <- function() {
  require("pkg1") # Replace pkg1 with the
                  # name of the desired package.
  require("pkg2")
  # and so on.
}
```



Indeed, the functions `.First()` and `.Last()` specify instructions to be executed when you start and exit R, respectively.

These functions can be put in a file called `.Renviron` located in the current folder or in the user's home directory given by the R instruction `Sys.getenv("R_USER")`.

References

1. Adler J. *R in a Nutshell*. O'Reilly, 2010.
2. Akaike H. Information Theory and an Extension of the Maximum Likelihood Principle. In Petrov B. N. and Csaki F., editors, *Proc. of the 2nd Int. Symp. on Information Theory*, pages 267–81, 1973.
3. Becker R. A., Chambers J. M. and Wilks A. R. *The New S Language: A Programming Environment for Data Analysis and Graphics*. Chapman & Hall, 1988.
4. Belsley D. A., Kuh E. and Welsch R. E. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York-Chichester-Brisbane, 1980.
5. Bilodeau M. and Lafaye de Micheaux P. A-dependence Statistics for Mutual and Serial Independence of Categorical Variables. *Journal of Statistical Planning and Inference*, 139:2407–19, 2009.
6. Braun J. W. and Murdoch D. J. *A First Course in Statistical Programming with R*. Cambridge University Press, 1st edition, January 2008.
7. Burns P. *The R inferno*. Unpublished manual, Amazon, 2011.
8. Chambers J. M. *Software for Data Analysis: Programming with R*. Statistics and Computing. Springer, June 2008.
9. Chivers I. and Sleightholme J. *Introduction to Programming with Fortran: With Coverage of Fortran 90, 95, 2003, 2008 and 77*. Springer, 2nd edition, 2012.
10. Cohen Y. and Cohen J. *Statistics and Data with R: An Applied Approach Through Examples*. Wiley, 2008.
11. Cook R. D. and Weisberg S. *Residuals and Influence in Regression*. Monographs on Statistics and Applied Probability. Chapman & Hall, London, 1982.
12. Crawley M. J. *The R Book*. Wiley, Chichester, June 2007.
13. Dalgaard P. *Introductory Statistics with R (Statistics and Computing)*. Springer, 2nd edition, August 2008.
14. Davison A. C. and Hinkley D. V. *Bootstrap Methods and their Applications*, volume 1 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 1997. With 1 IBM-PC floppy disk (3.5 inch; HD).
15. Everitt B. S. and Hothorn T. *A Handbook of Statistical Analyses Using R*. Chapman & Hall/CRC, 1st edition, February 2006.
16. Fisher R. A. *Statistical Methods for Research Workers*, 4th edition §21.1. Oliver & Boyd, Edinburgh, 1932.
17. Fox J. Extending the R Commander by “plug in” Packages. *R News*, 7(3):46–52, 2007.
18. Furnival G. M. and Jr. Wilson R. W. Regression by Leaps and Bounds. *Technometrics*, 16:499–511, 1974.
19. Hand D. J. Branch and Bounds in Statistical Data Analysis. *The Statistician*, 30:1–13, 1981.
20. Heiberger R. M. and Neuwirth E. *R Through Excel*. Use R! Springer, 2009.

21. Ihaka R. and Gentleman R. R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
22. Kernighan B. W. and Ritchie D. M. *The C Programming Language, Second Edition*. Prentice Hall, 1988.
23. Lafaye de Micheaux P. and Liquet B. ConvergenceConcepts: An R Package to Investigate Various Modes of Convergence. *R Journal*, 1(2):18–26, 2009.
24. Lafaye de Micheaux P. and Liquet B. Understanding Convergence Concepts: A Visual-minded and Graphical Simulation-based Approach. *The American Statistician*, 63(2):173–8, 2009.
25. Levene H. Robust Tests for Equality of Variances. In *Contributions to probability and statistics*, pages 278–92. Stanford Univ. Press, Stanford, Calif., 1960.
26. Maindonald J. and Braun J. *Data Analysis and Graphics Using R: An Example-based Approach (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge University Press, December 2006.
27. Mallows C. L. Some Comments on c_p . *Technometrics*, 15:661–75, 1973.
28. Matloff N. *The Art of R Programming*. No Starch Press, 2011.
29. Meyer C. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. With 1 CD-ROM (Windows, Macintosh and UNIX) and a solutions manual (iv+171 pp.).
30. Jr. Miller and Rupert G. *Simultaneous Statistical Inference*. Springer Series in Statistics. Springer-Verlag, New York, 2nd edition, 1981.
31. Montgomery D. C. *Design and Analysis of Experiments*. Wiley, 7th edition, 2008.
32. Muenchen R. A. *R for SAS and SPSS Users*. Springer Series in Statistics and Computing. Springer, 2009.
33. Muenchen R. A. and Hilbe J. M. *R for Stata Users*. Statistics and Computing. Springer, 2010.
34. Park S. K. and Miller K. W. Random Number Generators: Good Ones are Hard to Find. *Communication ACM*, 31(10):1192–1201, 1988.
35. Robert C. and Casella G. *Introducing Monte Carlo Methods with R (Use R!)*. Springer, 2009.
36. Sarkar D. *Lattice: Multivariate Data Visualization with R. Use R!* Springer, March 2008.
37. Schwarz G. Estimating the Dimension of a Model. *Annals of statistics*, 6:461–64, 1978.
38. Stroustrup B. *The C++ Programming Language, Third edition*. Addison-Wesley, 1997.
39. Teeter P. R *Cookbook*. O'Reilly, 2011.
40. Vinod H. D. *Hands-on Intermediate Econometrics Using R: Templates for Extending Dozens of Practical Examples*. World Scientific, Hackensack, NJ, 2008.
41. Weisberg S. *Applied Linear Regression, 3rd edition*. Wiley-Interscience, 2005.
42. Zuur A. F., Ieno E. N. and Meesters E. H.W.G. *A Beginner's Guide to R*. Springer, 2009.

General Index

A

- absolute deviation to the median 351
- absolute value 314
- AIC 488
- analysis of variance
 - one-way 503
 - one-way with repeated measures 522
 - repeated measures 521
 - two factors 513
 - two factors with repeated measures
 - for both factors 523
 - two factors with repeated measures
 - for one factor 525
- ANCOVA 468
- AND 98
- ANOVA 468
- arccosine 314
- arcsine 314
- arctangent 314
- arrays 52, 106
 - extraction 106
 - insertion 106
- arrow 283
- arrow, assignment 39
- assignment 39
- assignment arrow 39
- asymmetry, coefficient 351

B

- BATCH 302
- Bernoulli 411
- beta 407, 411
 - function 314
 - logarithm of beta function 314
- between-subjects factor 522
- bias estimated by bootstrap 404
- bias of an estimator 400

- BIC 483, 488
- binary 50, 71, 127, 191
- binomial 405
- binomial coefficient 314
- bit 128
- Bonferroni 510
- boolean 48
- bootstrap 396, 404
- boxplot 365
- bytes 50, 191

C

- carriage return 172
- Cauchy 407, 411
- central limit theorem 393
- character strings 49, 73, 108
 - case 111
 - concatenate 110
 - letters 110
 - lower case 111
 - replacing occurrences 110
 - search 110
 - split 110
 - sub-strings 110
 - upper case 111
- charts
 - bar 359, 363
 - bar with CF line 362
 - bar, stacked 361
 - boxplots 365, 368, 375
 - cross 357, 363
 - mosaic 372
 - Pareto 360
 - pie 361
 - stacked bar 361
 - stemplot 365, 367
- chi-squared 407
- chi-squared, Pearson's 352

- clipboard 69
- closing the session 38
- coefficient of determination 459, 471
- coefficient of variation 351
- Cohen-Friendly association plot 373
- collinearity 481
- colours 163
- command history 287
- comment 38
- compilers 231
- complex numbers 47
 - argument 47
 - imaginary part 47
 - modulus 47
 - real part 47
- concatenation 73
- condition 116
- confidence intervals 418, 461
 - for a correlation 423
 - for a mean 418
 - for a median 422
 - for a proportion 419
 - for a variance 421
- confidence intervals, interpretation 449
- console 37
- contingency coefficient, Pearson's 353
- contingency table 352
- contrasts, in ANOVA 510, 512, 519
- control flow 115
- Cook's distance 493
- copy-pasting 69
- correlation 417
- correlation ratio eta-squared 356
- cosine 314
- counter 119
- covariance ratio 495
- Cramér, Phi-2 and V-2 353
- CRAN 5
- creating data 73
- cumulative distribution function 386, 389
- cumulative distribution function, empirical 363
- cumulative frequency polygon .. 349, 370
- cumulative maxima 314
- cumulative minima 314
- cumulative products 314
- cumulative sums 314
- D**
- databases 76
- datasets 145
- dates 111, 138
 - anteriority test 115
 - current date 111
 - day 111, 113
 - difference 115
 - extraction 112
 - hours 111, 113
 - minutes 111, 113
 - month 111, 113
 - operations on 115
 - POSIX 138
 - seconds 111, 113
 - time zone 113
 - year 111, 113
- debugger 257
- debugging 255
- density 386, 389
- Dfbetas 495
- Dffits 494
- directory 285
- distribution function 386
- distribution function, empirical 363
- DLL 238
- E**
- effective argument 194
- elements
 - extraction 99
 - insertion 99
- elements
 - replacement 101
- empirical cumulative distribution function 396
- empirical distribution function .. 367
 - plot 363, 367
- environment 228

estimator 394
 Excel 69, 72
 execution time 112
 exit 38
 exponential 314, 407, 411
 exporting data 72
 expression 224

F

factorial 314
 factors 56
 factors, in ANOVA 503
 FALSE 97
 FAQ 143, 147
 file path 64, 65
 Fisher 407
 fixed effects model 521
 fonts 180
 formal arguments 195
 format 278
 formulae 226, 505
 fractiles 350, 389
 frequency polygons 369
 functions
 arguments 44
 body 195
 comments 195
 constrained optimization 330
 declaration 194
 demonstration 145
 errors 124
 examples of use 145
 methods 144
 multidimensional optimization 328
 naming of effective arguments 196
 object oriented programming 204
 operators 202
 optimization 327
 parameters 43
 partial naming of effective arguments 196
 returning an object 198
 roots 331
 scope of variables 200
 sequence of instructions 195

supplementary arguments 197
 variable scope 200
 zeros 331

G

gamma
 distribution 407, 408
 first derivative of logarithm of gamma function 314
 function 314
 logarithm of gamma function 314
 second derivative of logarithm of gamma function 314
 generalized error distribution 408
 generalized Pareto 408, 414
 generating numbers 405
 generating random numbers 381
 geometric 405
 gradient, numerical 327
 graphical user interface 7
 graphics window 151
 font size 152
 height 152
 managing parameters 177
 splitting 153
 width 152
 Gumbel 407

H

help
 online 141
 hexadecimal 50
 histogram 368
 history 287
 hyperbolic cosine 314
 hyperbolic sine 314
 hyperbolic tangent 314
 hypergeometric 405
 hypothesis testing 424
 assertion of interest 424
 Bartlett 431, 509
 chi squared for fit to a distribution 440
 decision rule 424
 Fisher's test in ANOVA 507
 Fisher's test, exact 438

for a correlation	433
for a mean	426
for a proportion	431
for a variance	429
for two correlations	434
for two means	427
for two means, paired samples	428
for two variances	430
independence, chi-squared	435
Kolmogorov-Smirnov for one sample	440
Kolmogorov-Smirnov for two samples	441
level of significance	424
Levene	509
Mann-Whitney for two samples	445
mutual independence for contingency tables	437
normality, Shapiro-Wilk	439
of sign, of median for two paired samples	444
of sign, of median for two samples	443
of sign, of the median for one sample	442
p-value	425
power	425, 451
risk of the first kind	424, 451
test statistic	424
two proportions	432
Wilcoxon	429
Wilcoxon for paired samples	446
Wilcoxon for two samples	445
Yates' chi-squared	437
I	
images	166
importing data	70
index of largest value	101
index of smallest value	101
indexing	
recursive	108
inference	394
inferential	410
influential point	491
inheriting classes	213
installation	
of R	531
packages	532
integer part	314
integration, numerical	325
interaction	478, 517
interquartile range	351
inverse hyperbolic cosine	314
inverse hyperbolic sine	314
inverse hyperbolic tangent	314
inverse normal	408
IRC	147
Irwin-Hall	408
J	
Johnson SB	408
Johnson SU	408
K	
Kendall's tau	354
KumaraSwamy	408
kurtosis	351
L	
Laplace	408
Laplace test	315
law of large numbers	392
least squares criterion	458
levels of a factor in ANOVA	503
leverage	492
Lévy	408
lists	53, 96, 106
extraction	106
insertion	108
loading a package	536
location contaminated	408
log-logistic	408
logarithm	314
logarithmic scale	157
logical	97, 116
logical mask	97, 100, 102, 105
logistic	407
lognormal	407
loop	118

M	
mailing lists	146
Mallows	483
mask	
logical mask	97
mass function	389
Matlab	70
matrices	52, 85, 88, 315
adding a scalar	316
addition, entry-wise	316
adjoint	322
centred and/or scaled	321
Cholesky decomposition	323
condition number	321
conjugation	317
cross product	317
determinant	320
division	317
division, entry-wise	317
eigenvalues	321
eigenvectors	321
extracting by indices	102
extracting by logical mask	102
extraction	102
half vec operator	320
Hermitian	322
insertion	105
inverse, Cholesky method	324
inversion	317
Kronecker product	319
lower triangular	319
Moore-Penrose	323
multiplication	317
multiplication, entry-wise	317
multiplying by a scalar	316
number of columns	88
number of rows	88
outer product	318
pseudo-inverse	323
QR decomposition	324
rank	325
singular value decomposition	323
singular values	323
singular vectors	323
square root	322
subtraction, entrywise	316
trace	320
transpose	322
transposition	316, 322
triangular	319
upper triangular	319
vec operator	320
maximum	314
maximum likelihood	397
mean	350, 417
mean absolute deviation	351
median	348, 417
merging columns	89
merging rows	89
merging tables	89
method	207
minimum	314
Minitab	70
missing values	48, 137
mixed effects model	521
mode	348
modelling	410
Monte Carlo	396
mosaic plot	372
multinomial	408
multiple comparisons, in ANOVA	
510	
MySQL	72, 76
N	
names of variables	40
natural logarithm	314
negative binomial	405, 411
normal	407
numbers	
binary notation	127
decimal notation	127
dyadic	128
fixed-point	127
floating point	127
numerical differentiation	327
numerical gradient	327
numerical integration	325
numerical optimization	327
numerical summaries	347

O

- object-oriented programming ... 193
- objects
 - attributes 216
 - class 205
 - deleting 284
 - listing 284
 - methods 206
 - storage 283
- online help 141
- OpenOffice 67, 69, 72
- optimization
 - constrained 330
 - multidimensional 328
- optimization, numerical 327
- OR 98
- outliers 491
- overflow 132

P

- p-value 450, 459
- packages 144, 283, 290, 303
- Pareto 411
- path 290
- peakedness, coefficient 351
- Pearson's chi-squared 352
- Pearson's contingency coefficient 353
- Pearson's correlation 355
- pi, number 315
- plots
 - 3D 187
 - adding text 169
 - adding text in the margin 170
 - arrows 160
 - axes 172, 183
 - boxes 162
 - caption 173
 - colours 163, 179
 - curves 162
 - drawing 156
 - fine-tuning 176
 - gif 169
 - identifying the points 175
 - interacting 175
 - legend 173

- lines 158
- lines and symbols 185
- logarithmic scale 157
- managing text 180
- overlay 157
- polygons 161
- saving 152
- segments 158
- straight line 159
- text 169
- title 157, 171
- Poisson 405, 411
- polynomial regression 496
- POSIXlt 112
- prediction 461
- prediction intervals 461
- predictor 461
- principal components analysis .. 335
- probability density function 386
- product 314
- prompt symbol 37
- proportion 417
- pseudorandom numbers 382

Q

- quantile function 386, 389
- quantiles 389
 - chi-squared 418
 - Fisher 418
 - normal 418
 - Student 418
- quotation marks 109

R

- Rademacher 408
- random effects model 521
- random variables 383
 - distribution 383, 385, 389
 - i.i.d. 384
 - identically distributed 384
 - independent 384
 - parameters 391
 - parameters of a distribution... 386
 - realizations 383, 384
 - support 387, 389

range 314, 351
 ranks 87
 Rayleigh 408
 RCommander 7
 recursive indexing 108
 recycling 86
 regular expressions 284
 rejection sampling 402
 remainder of division 314
 residuals 463, 490
 Rice 408
 roots of a polynomial 332
 rounding 314

S

sample 385
 sampling variation 399
 SAS 70, 72
 saving 283, 285, 288, 292
 scale contaminated 408
 search engines 145
 sets
 complement 99
 inclusion 99
 intersection 99
 membership 99
 subset 99
 superset 99
 symmetric difference 99
 union 99
 shifted exponential 408
 sign 314
 simulation 381
 sine 314
 skew-normal 408
 skewness 351
 Spearman's rank correlation coefficient 355
 spreadsheet 75
 SPSS 70
 square root 314
 stable 408
 standard deviation 351
 standardized residuals 491
 stemplot 365, 367

strings of characters 49, 108
 Student 407
 studentized residuals 492
 subject factor 522
 sum 314
 symbolic differentiation 326
 symmetrical Tukey 408

T

tables
 conditional distributions 346
 contingency 344, 352
 contributions to the chi-squared 352
 counts 343
 frequency 343
 grouped data 344
 independence 352
 individual data 343
 joint distribution 345
 marginal distributions 346
 margins 345
 tangent 314
 Task Views 147
 text editor 63
 tie 87
 TRUE 97
 Tukey 511

U

underflow 132
 uniform 382, 407
 discrete 405

V

variable name 40
 variables
 dummy 474
 explained, dependent, response 455
 explanatory, independent 455
 independent, in ANOVA 513
 interaction 478
 modalities 340
 mode 348
 ordinal 341, 342
 qualitative 341

quantitative, continuous	341, 343
quantitative, discrete	341, 342
type	340
variance	351, 417
variance estimated by bootstrap	404
variance inflation factor	481
variance of an estimator	400
vectorization	86
vectors	51, 85
duplicates	88
length of a vector	87
ranking indices	87
sorting elements	87
vector of ranks	87
VIF	481
vignettes	145
W	
Weibull	407
Welsch-Kuh distance	494
<i>wiki</i>	147
window	
graphics	151
workspace	285
X	
XOR	98

Index of R Commands and Symbols

Symbols	
"	109
~	187, 206, 226, 276
+	86, 203
-	203
->	59, 283
.	197
.C()	238, 239, 242, 253
.Call()	253
.First()	538
.Fortran()	238, 242
.GlobalEnv	228, 229
.Last()	538
.Machine	133
/	203
:()	569
;	43, 195
<	98, 134, 203
<-	39, 59, 101, 105, 108, 204, 229, 283
<=	98, 134, 203
=	39, 195
==	98, 118, 134, 203
>	37, 98, 134
>=	98, 134, 203
?	141, 149
??	144
[85, 100, 102, 106, 134, 135
[()	104
[[85, 107, 134, 135
#	38, 195
\$	108
%/%	203
%%	203, 314
%in%	99, 203, 526
%o%	203
%*%	317, 333
%*%()	575
&	98, 134, 203
&&	98, 117, 134
	98, 203
	98
!	98, 203
!=	98, 203
:	51, 52, 73, 206, 496
^	203
'	49
{	195
}	195
*	203, 479
A	
A.dep.tests()	437
abline()	26, 159, 181, 186, 188, 371, 458, 468, 479, 480, 571
abs()	314, 333, 351
acos()	314
acosh()	314
add	162
add1()	484-486, 577
addmargins()	345, 377
ade4	337, 373
adj	180
aggregate()	95
all()	91, 98, 99, 134, 135, 137
all.equal()	97, 98, 118, 131, 323, 324, 335, 379
along.with	196
alpha	165
alternative	427, 432, 433, 438, 442
AnalyzeFMRI	191, 594
ann	180
Anova()	507, 516, 517, 521, 527

anova() ... 460, 472, 474, 477, 496,
 506, 513, 516, 517, 521, 577
anova(lm()) 497
ansari.test() 448
any() 98, 134, 135, 137
aov() . 505, 506, 511, 513, 516-518,
 521, 523, 525-527, 578
aplypack 365, 367
apply() .. 93, 94, 96, 120, 134, 137,
 190, 191, 415, 569
approx() 158
apropos() 144, 149, 204
Arg() 47
arima() 241
arima.sim() 241
array() 52, 58, 59, 106, 220
arrowaxis() 357
arrows() 160, 161, 188
as.character() . 50, 109, 207, 238
as.data.frame() 68, 83
as.Date() 57, 58
as.double() 238, 343, 377
as.factor() 341, 377, 527
as.integer() ... 50, 238, 342, 377
as.list() 201
as.logical() 98, 137, 341
as.matrix() 336
as.numeric() 50, 191, 348
as.ordered() 342, 377
as.POSIXct() 113, 114
as.POSIXlt() .. 113-115, 134, 138
as.raw() 50
as.vector() 103, 104, 190
asin() 314
asinh() 314
ask 177
assocplot() 373
asymp.test() . 422, 430, 431, 447,
 448
asympTest 422, 430
at 173
atan() 314
atanh() 314
attach() 66, 81, 135, 287, 291,
 307, 340, 378, 456, 569, 600

attr() 205, 217-219, 222, 276
attributes() .. 216-219, 222, 223,
 276, 327
axes 162, 172
axis 172
axis() 172, 173, 184, 188, 192
axTicks() 183

B
barchart() 359, 360, 362
barlett.test() 513, 527
barplot() . 151, 190, 359-363, 372,
 377, 576
bartlett.test() ... 448, 509, 578
basename() 292
BATCH 302
beta() 314
bg 179, 186
bigmemory 301
bin2dec() 128
binom 449
binom.approx() 420
binom.exact() 421
binom.test() . 420, 421, 424, 432,
 442, 444, 445, 447-449, 577
biroot() 122
bitmap() 153, 289
BMI() 121
bmp 152
bmp() 153
boot() 405, 419, 421-423, 449, 577
boot.ci() 419, 421-423
box() 162, 163, 171, 185, 188
Box.test() 448
boxplot() .. 27, 365, 366, 368, 377,
 576
bquote() 169
break 118-120, 126
breaks 344
Brodbdingnag 132
browseEnv() 601
browser() 255-257
bty 162, 183
by 90, 92, 95, 196
byrow 52

C	
c.....	257
C()	508, 519, 527, 578
c()	51, 52, 58, 59, 73, 74, 80, 101, 198, 569
camembert()	361
car	482, 507, 509, 516, 533
cat()	109, 119, 194, 196, 198, 207
caTools	169, 189
cbind()	89, 90, 104, 134, 190
ceiling()	314
cex.....	177, 180
cex.axis	180
cex.lab.....	180
cex.main	180
cex.sub.....	180
character	49, 50
chisq.test()	352, 377, 436-438, 440, 443, 447-449, 577
chol()	323, 324, 333
chol2inv()	324
choose()	46, 232, 314, 575
choose.dir()	307, 574
choose.files()	535
chron	115
cin	180
class()	46, 50, 51, 56, 59, 107, 205, 206, 212, 214, 217, 218, 220, 223, 276, 378, 506
clip()	177
CMD	535
co.var()	351
coeff()	479, 480
coefficients()	460, 577
col ...	163, 164, 173, 179, 186, 190, 571
col.axis	179
col.lab.....	179
col.main	179
col.names	223
col.sub.....	179
col2rgb()	165
colClasses	568
collapse.....	135, 198-200
collapse()	201, 207
colMeans()	81, 93, 336
colnames	68, 223
colnames()	88, 134, 223, 569
colors()	163, 188
colSums()	93
combinat	232
combn()	231, 232
combnRC()	239, 257
Commander()	18
complete.cases()	137
complex	47, 50
conf.level	419, 421
confint()	461, 468, 473, 481, 496, 497
constrOptim()	327, 331
contour()	192
contour3d()	22
ConvergenceConcepts	393
cooks.distance()	493, 494, 497
cor()	337, 354, 355, 377, 417
cor.test()	356, 377, 423, 424, 434, 447-449
cor.test.2.sample()	435, 447, 448
cor0.test()	434, 448
correct	436, 447
cos()	165, 314, 333
cosh()	314
cov()	336, 355, 377
covratio()	495
cra	180
cramer.v()	353
crosschart()	357, 363, 367
crossprod()	317
CrossTable()	439
crt	180
csi	177, 180
cummax()	314, 333
cummin()	314, 333
cumprod()	314, 333
cumsum()	314, 333
curve()	26, 162, 165, 179, 188, 189, 571
cut()	56, 344

cxy 180

D

D() 326, 333
 data() 145, 149, 570
 data.entry() 80, 569
 data.frame() 55, 58, 59,
 90, 93, 95, 111, 135, 140, 223,
 462, 473
 Date 115
 date() 111
 dbeta() 407
 dbinom() 405
 dcauchy() 407
 dchisq() 407
 de() 75, 80, 81, 569
 debug() 257
 dec 64, 80, 568
 dec2bin() 128
 demo() 145, 149, 170, 182, 187
 density() 390
 deriv() 326, 333
 det() 320, 333, 334, 575
 detach() 291, 307, 600
 detectCores() 274
 dev.copy2eps() 153
 dev.copy2pdf() 153
 dev.cur() 152
 dev.list() 152
 dev.off() 152, 153, 188, 289, 571
 dev.print() 289
 dev.set() 152
 devAskNewPage() 177
 device 152
 dexp() 407
 df() 407, 576
 dfbetas() 495, 497
 dffits() 494, 497
 dgamma() 407
 dgeom() 405
 dgumbel() 407
 dhyper() 405
 diag() 315, 320, 322, 575
 diff() 351, 576
 difftime() 115, 134

digamma() 314
 dim 53, 217, 219, 220
 dim() 88, 134, 219, 220
 dimnames 217, 221-223
 dimnames() 88, 134, 223
 din 177
 dir.create() 292
 display.brewer.all() 167
 dlnorm() 407
 dlogis() 407
 dnbinom() 405
 dnorm() 162, 407, 413
 do.call() 97, 139, 140
 dotchart() 357, 358
 double 46
 download.file() 536
 dpois() 405
 dQuote() 109
 drop 104
 drop1() 486, 487, 577
 dt() 407
 dudi.pca() 337
 dunif() 407
 duplicated() 88
 dweibull() 407
 dwtest() 464
 dyn.load() 238, 261
 dyn.unload() 238

E

ecdf() 363, 364, 367, 370, 371, 397
 edit 69
 eigen() 321, 322, 333, 334, 575
 else 116, 134
 emptyenv() 229
 epitools 420, 421
 Error() 523, 525-527
 estimable() 519, 520, 527
 eta2() 356
 eval() 224, 225, 276
 evd 407
 exact 446, 447
 example() 143, 145, 149, 161, 187,
 365
 exp() 43, 314, 333

F
 expression() .. 169, 170, 224-226,
 276, 326

F
 F 48, 97
 f.read.volume() 191
 factor() .56, 58, 59, 278, 476, 506,
 513, 521, 527, 577
 factorial() 45, 314
 FALSE 48, 50, 59, 91, 97
 family 180
 ff 301
 fg 179
 fig 177
 file 64, 65, 80
 file(clipboard) 80
 file.access() 292
 file.append() 292
 file.choose() 64, 80, 256
 file.copy() 292
 file.create() 292
 file.exists() 278, 292
 file.info() 292
 file.path() 278, 292
 file.remove() 292
 file.rename() 292
 file.show() 292
 file.symlink() 292
 filehash 72
 filename 152
 fill 173
 fin 177
 find() 144
 fisher.test() 438, 443, 444,
 447-449, 577
 fit.contrast() 512, 513, 527
 fitted() 465, 492
 fix() 69, 75, 80, 568
 flashy.plot() 374
 fligner.test() 448
 floor() 314
 flush.console() 119
 font 180
 font.axis 180
 font.lab 180

font.main 180
 font.sub 180
 for 118, 119, 134
 force.in 483
 foreign 70, 71, 568, 581
 format() 109, 113
 formatC() 130
 formula 206, 207, 212
 formula() 212
 fourthrows() 140
 freq 369
 friedman.test() 448
 ftable() 80, 345
 FUN 93
 function print() 208
 function() .. 26, 45, 194, 196-201,
 206, 207, 276

G
 gamma() 314
 gc() 299-301
 gdata 70, 93, 568, 583, 587, 600
 get() 307
 getaddr() 296
 getAnywhere() 213
 getLoadDLLs() 255
 getwd() 285, 574
 ggplot2 187
 gl() 57, 514, 523, 525, 526
 glob2rx() 284
 globalenv() 229, 230
 gmodels 439, 519, 527
 gputools 274
 grad() 327, 333
 graphics.off() 152
 gray() 191
 gregmisc 512, 513, 527
 grep() 110, 134, 140, 163
 gsub() 110, 134
 gtools 92

H
 hat() 169, 493
 hatvalues() 493
 head() 65, 340, 378

header 64, 80, 568
height 152
heights 156
help() 42, 141-144, 149, 570
help.search() 144, 149
help.start() 143, 144, 149
hessian() 327, 333
hist() 27, 151, 153, 162, 344, 349,
 368-370, 377, 464, 576
history() 288, 574

I

I() 496
identical() 98, 220
identify() 175, 188, 571
if() 116, 118, 134
ifelse() 117
Im() 47
image() 166, 168, 169, 189, 191,
 192, 571
IndependenceTests 437
Inf 49
influence.measures() 493
inherits() 206, 214
INSTALL 535
install.packages() 535
integer 46, 52
integrate() 325, 333, 335, 413
interaction.plot() 515, 516,
 521, 527
intersect() 99
inv() 248
invisible() 200, 201
IQR() 351, 377, 576
is.character() 49, 59
is.element() 99
is.list() 55
is.loaded() 255
is.logical() 59, 98
is.na() 49, 59, 137
is.null() 125
is.numeric() 59
isTRUE() 118, 131

J

jarque.bera.test() 463, 465

jpeg 152
jpeg() 153, 289
jpg 152

K

kappa() 321
kde2d() 415
Kendall.taub() 354
kronecker() 319, 333
kruskal.test() 448
ks.test() 441, 442, 447-449, 577
kurt() 352
kurtosis() 352

L

lab 183
labels 173
lapply() 96, 134
las 183
lattice 187
layout() 153-156, 177, 188, 192,
 571
layout.show() 155, 156
lbeta() 314
leaps() 483
legend 173
legend() 173, 174, 188, 192, 463,
 479, 480
len 183
lend 185
length() 87, 134
length.out 196
LETTERS 92, 110
letters 51, 92, 110, 222
levels 56
levels() 56, 341, 342, 344, 377,
 378
levene.test() 509, 513, 527, 578
lgamma() 314
lheight 185
library() 144, 149, 290, 574
lines() 158, 159, 173, 185, 188,
 227
list() 53, 54, 58, 59, 195, 197, 201,
 222
list.files() 292

ljoin 185
 lm() 26, 213, 458,
 459, 461, 468, 471, 474, 476,
 478, 481, 482, 485-487, 489,
 490, 496, 497, 506-508, 513,
 516-519, 521, 527, 577
 lme() 523
 lmitre 185
 lmtest 464
 load() 278, 285, 287
 loadhistory() 287
 local() 229, 230, 276
 locator() 175, 188, 189, 371, 571
 log 157, 183
 log() 43, 45, 314, 333
 logical() 48, 50, 98
 lower.tri() 319
 ls() 75, 229, 284, 291, 307, 574
 lty 162, 173, 185, 186
 lwd 173, 185, 186

M

m 367
 mad() 351, 377
 mai 177
 main 157, 171
 mantelhaen.test() 448
 map() 191
 mapdata 191, 594
 mapply() 97
 maps 191, 594
 mar 177
 MARGIN 93
 margin.table() 346, 377, 575
 MASS 415
 mat 71
 match() 203
 match.call() 201
 matlines() 463
 Matrix 316
 matrix() 52, 58, 59, 134, 168, 191,
 219, 220, 223, 315, 319, 361
 mauchley.test() 448
 mauchly.test() 448
 max() 314, 333, 351

mcnemar.test() 448
 mean() 81, 142, 149, 350, 351, 377,
 417
 med.test() 449
 median() 315, 348, 377, 417, 609
 memory.limit() 301
 memory.size() 301
 meresix() 140
 merge() 90, 92, 134
 method 483
 methods() 144, 211, 212
 mex 177
 mfcol 154, 177
 mfg 177
 mfrow 153, 177
 mgp 177
 min() 314, 333, 351
 misc3d 22
 missing() 196, 198, 201, 276
 Mod() 47
 mode() 46, 49, 50, 59
 model.frame() 214
 model.matrix() 493, 508
 moments 352, 576
 months() 113
 mood.test() 448
 mosaicplot() 372
 mpinv() 323
 mtext() 170, 180, 184, 188
 mtp 71
 mtrace() 257, 258
 mvtnorm 253

N

n 256
 NA 48-50, 59, 91, 97, 377
 na.omit() 137, 377
 na.rm 139
 names 217, 221-223
 names() 51, 88, 138, 201, 222, 223,
 460, 575
 NAMESPACE 261
 NaN 49
 nchar() 109, 134, 135
 ncol 52

ncol() 88, 134
 ncolumns 72
 new 177
 new.env() 229, 230, 276
 next 118, 119
 nlevels() 344
 nlm() 327-330, 333
 nlme() 523
 nlminb() 327, 330, 331, 333, 398
 noquote() 109
 nortest 448
 nrow 52
 nrow() 88, 134
 nrows 80, 568
 NULL 116, 125, 198, 218, 220
 numDeriv 327, 333
 numeric 46, 50
 numericDeriv() 327

O

object.size() 298, 301
 objects() 284, 574
 odbcClose() 76
 odbcConnect() 76
 oma 177
 oma 177
 oma 177
 oneway.test() 448
 optim() 327
 optimize() 327, 328, 333, 575
 options() 144
 order() 87, 134, 139, 161
 ordered() 56, 58
 origin 114
 ormidp.test() 448
 outer() 318, 330, 333

P

package.skeleton() 304, 308
 paired 428, 446, 447
 pairs() 470, 496, 497
 pairwise.prop.test() 448
 pairwise.t.test() 448, 510, 513,
 527, 578
 pairwise.wilcox.test() 448

par() 27, 153, 154, 164, 176, 177,
 179-185, 188, 189, 192, 571
 parallel 273, 274
 parent.env() 230
 parse() 225, 276
 paste() 110, 112, 134, 135, 191,
 198-201, 203, 207, 223
 paste(..., sep = , collapse
 = NULL) 198
 path.expand() 292
 pattern 284
 pbeta() 407
 pbinom() 405
 pcauchy() 407
 pch 185, 186, 357
 pchisq() 407, 413
 pdf 152
 pdf() 153, 289
 permn() 232
 persp() 329, 330
 pexp() 407
 pf() 407
 pgamma() 407
 pgeom() 405
 pgumbel() 407
 phyper() 405
 pi 315
 pictex() 153
 pie() 166, 361, 377, 576
 pin 177
 plnorm() 407
 plogis() 407
 plot() 26, 156, 157, 159-164,
 169-172, 174, 175, 181-184,
 186-189, 357, 363, 364, 367,
 370, 371, 374-377, 457, 479,
 480, 513, 516
 plot(lm()) 497
 plot.default() 183
 plot.ecdf() 377
 plot.lm() 215
 plot.new() 171, 173, 177, 185, 279
 plot.window() 279
 plotMeans() 515
 plt 177

plyr 96
 pmvtnorm() 253
 pnbinom() 405
 png 152
 png() 153, 289, 574
 pnorm() 335, 407, 449, 577
 points() 26, 156-158, 186, 188,
 227, 362, 479, 480, 571
 pointsize 152
 poly() 496
 polygon() 161, 188, 189
 polyroot() 122, 332-334, 575
 pos 170
 POSIXct 113
 POSIXlt 113, 138
 postscript() 153, 289
 PoweR 442
 power.anova.test() 448
 power.prop.test() 448
 power.t.test() 448
 PP.test() 448
 ppois() 405
 predict() 462, 463, 468, 473, 496,
 497
 print 209
 print() 109, 197, 201, 204,
 206-208, 210, 226
 print.default() 208, 211, 214
 print.formula() 207
 print.lm() 214
 PrintValue() 261
 prod() 137, 314, 333, 569
 prop.table() 346, 377, 575
 prop.test() 420, 424, 431-433,
 442-444, 447-449, 577
 prop.trend.test() 448
 ps 152, 180
 pt() 407
 pty 177
 punif() 407
 pweibull() 407
 pwilcox() 445

Q

Q 256

q() 38, 287
 qbeta() 407
 qbinom() 405, 422, 423, 577
 qcauchy() 407
 qchisq() 407, 418, 576
 qexp() 407
 qf() 407, 413, 418
 qgamma() 407
 qgeom() 405
 qgumbel() 407
 qhyper() 405
 qlnorm() 407
 qlogis() 407
 qnbinom() 405
 qnorm() 407, 418
 qpois() 405
 qqnorm() 464
 qr() 324, 333
 qr.Q() 324
 qr.R() 324
 qt() 407, 413, 418, 576
 quade.test() 448
 quantile() 350, 377
 quartz() 152
 qunif() 407
 quote() 327
 qweibull() 407

R

R.matlab 4, 71, 582
 R2HTML 290
 rainbow() 165, 166, 192
 range() 314, 351, 377, 575, 576
 rank() 87, 355
 rate2by2.test() 448
 raw 50
 rbenchmark 242
 rbeta() 407
 rbind() 89, 92, 134, 139
 rbinom() 403, 405, 411
 rcauchy() 407
 rchisq() 407
 Rcmdr 7, 8, 515, 533, 535-538
 RcmdrPlugin.sos 19
 RcmdrPlugin.TeachingDemos 18

RColorBrewer 166, 167
 RCommander 8, 12, 13, 16, 17
 Re() 47
 read.csv() 80, 568
 read.csv2() 80, 568
 read.delim() 80, 568
 read.delim2() 80, 568
 read.ftable() 64, 80, 345, 568
 read.gif() 169, 189
 read.mtp() 70, 71, 80
 read.spss() 70, 71, 80
 read.table() 64, 69, 71, 80-82,
 456, 568
 read.xls() 70, 340, 457, 568
 read.xport() 70, 71, 80
 readBin() 191
 readline() 24
 readLines() 80, 568
 readMat() 71, 80
 rect() 189
 regsubsets() 483, 484, 497
 relevel() 477
 rep() 73, 80, 166, 569
 repeat 118, 120, 134
 replicate() 385, 387, 415
 require() 167, 261, 290, 340, 536,
 574
 resid() 464
 residuals() 464, 465, 497, 577
 return() 121, 199
 rev() 87, 134, 335
 RevoScaleR 301
 RExcelInstaller 4
 rexp() 407, 411
 rf() 407
 Rf_PrintValue() 261, 264
 rgamma() 407
 rgb() 165
 rgeom() 405
 rgl 187, 280, 415
 rgrs 353
 rgumbel() 407
 rhyper() 405
 rlnorm() 407
 rlogis() 407

rm() 81, 82, 284, 301, 574
 Rmpi 273
 rnbinom() 405, 411
 rnorm() 74, 162, 392, 401, 407, 576
 RODBC 76
 round() 314, 333
 row.names 64, 80, 223, 568
 rowMeans() 93
 rownames 223
 rownames() 88, 134, 175, 223, 569
 rowSums() 93
 rpois() 405, 411
 RSiteSearch() 146, 149, 570
 rstandard() 491, 492, 497
 rstudent() 492, 497
 rt() 407
 runif() 74, 161, 163, 164, 383, 397,
 401-403, 407, 413
 rweibull() 407
 Ryacas 326

S

s.class() 337
 sample() 140, 404, 405, 415
 sapply() 96, 120, 134, 207
 sav 71
 save() 278, 285
 save.image() 285, 287
 savehistory() 287
 savePlot() 152, 153, 188, 571
 scale() 321, 336, 575
 scan() 64, 68, 69, 71, 74, 80, 82,
 192, 568
 scatter() 337
 sd() 321, 351, 377
 sd.pop() 351
 search() 290, 291, 307
 searchpaths() 290
 seed 382
 segments() 158, 159, 188, 369
 sep 64, 80, 198, 568
 seq() 51, 73, 80, 161, 196, 202, 569
 set.seed() 383
 setdiff() 99
 setwd() 65, 236, 285, 307, 574

shapiro.test() 439, 447-449, 577
 side..... 170, 173
 sigma2.test() 421, 424, 429, 430,
 447, 448
 sign() 314
 signif() 314
 sin() 43, 165, 314, 333
 sinh() 314
 sink() 292
 skew() 352
 skewness() 352
 skip..... 80, 568
 smartbind() 92
 solve() ... 248, 260, 317, 324, 333,
 575
 sort() 87, 134, 138
 source() 41, 42, 202, 256, 292
 spheres3d() 415
 spineplot() 373
 split.screen() 177
 sprintf() 109
 sqlQuery() 76
 sqrt() 43, 314, 321, 333
 sQuote() 109
 srt..... 180, 183
 stack() 94, 506
 stats..... 149
 stem() 367, 377
 stem.leaf() 365, 367
 step() 488, 489, 497
 stop() 123, 124
 storage.mode() 50
 str() 55
 strheight() 180, 185
 stripchart()..... 376
 strftime() 112-114, 134, 138, 570
 strsplit() 110, 134, 135
 strwidth() 180
 style..... 367
 sub..... 157, 171
 sub() 110, 134
 subset() 105, 517
 substring() ...110, 112, 134, 135,
 138, 139
 sum() 314, 320, 333

summary() ..83, 210, 215, 350, 377,
 436, 459, 468, 471, 481, 484,
 496, 507, 508, 513, 517-519,
 521, 523, 525, 526, 576, 578
 summary(lm()) 497
 summary.lm() 215
 surface3d() 415
 svd() 323, 333
 sweep() 94
 switch..... 134
 switch() 116, 125, 196
 symbol58 479
 Sys.chmod() 292
 Sys.getpid() 262
 Sys.sleep() 112, 119
 Sys.time() 111-113, 134, 278
 Sys.umask() 292
 system() 236
 system.time() 112, 120, 232

T

T 48, 97
 t() 83, 104, 190, 317, 324, 333, 575
 t.test() 419, 424, 426-428,
 447-450, 577
 tab2by2.test() 448
 table() ... 343, 344, 357, 377, 436,
 438, 444, 575, 576
 table.cont() 373, 374
 tabulate() 190
 tail() 65, 378
 tan() 314
 tanh() 314
 tapply() 120, 506
 tck 183
 tcl 183
 tcltk 7, 8
 test() 449
 text() 166, 168-170, 175, 180-182,
 184, 185, 188, 571
 TheRSoftware viii
 tick 173
 tiff() 153
 title() 171, 180, 185, 188
 tolower() 111, 570

toupper() 111, 222
 trace() 320
 tracemem() 216
 transform() 95, 111, 456
 trigamma() 314
 TRUE 48, 50, 59, 91, 97, 134
 trunc() 314
 try() 124
 ts() 57, 58
 tseries 463
 TukeyHSD() 511
 twentyfourthrows() 140
 type 157
 typeof() 46, 59

U

undebug() 257
 union() 99
 unique() 88, 134, 138, 415
 uniroot() 331, 333, 334, 575
 unit 367
 unlink() 292
 unlist() 139, 140
 unstack() 94
 update() 187, 212, 228
 upper.tri() 319
 useDynLib 261
 UseMethod() 207, 209, 211
 usr 177
 utils 591

V

var() 351, 377, 417
 var.equal 428, 447
 var.pop() 351
 var.test() 430, 431, 447-449
 vec() 320
 vech() 320
 vector 51
 vector() 126
 View() 69
 vif() 482, 496, 497
 vignette() 145, 149, 326

W

weekdays() 113

weight.category() 125
 which() 100, 104, 106, 134
 which.max() 100, 101, 575
 which.min() 100
 while() 118, 119, 126, 134
 width 152
 widths 156
 wilcox.test() 423, 424, 445-449,
 577
 win.graph() 151
 windows() 151, 152, 188, 571
 wmf 152
 write() 72, 569
 write.ftable() 82
 write.table() 72, 80, 81, 569
 writeaddr() 296

X

X11() 152
 xaxp 183
 xaxs 183
 xaxt 183
 xfig() 153
 xlab 157, 171
 xlim 157
 xlog 183
 xls 70
 xlsReadWrite 70, 73, 581
 xor() 98
 xpd 177
 xpinch 152
 xpos 152
 xpt 71
 xtable 145
 xyplot() 187

Y

yaxp 183
 yaxs 183
 yaxt 183
 ylab 157, 171
 ylim 157
 ylog 183
 ypinch 152
 ypos 152

Index of Authors

- Adler, J. 148
Akaike, H. 488
- Becker, R. A. 3
Belsley, D. A. 495
Bilodeau, M. 437
Braun, J. 148
Braun, J. W. 148
Burns, P. 132, 148
- Casella, G. 382
Chambers, J. M. 3, 148
Chivers, Ian 231
Cohen, J. 148
Cohen, Y. 148
Cook, R. D. 495
Crawley, M. J. 148
- Dalgaard, P. 148
Davison, A. C. 419
- Everitt, B. S. 148
- Fisher, R. A. 402
Fox, J. 18
Furnival, G. M. 483
- Gentleman, R. 3
- Hand, D. J. 483
Heiberger, R. M. 4
Hilbe, Joseph M. 4
- Hinkley, D. V. 419
Hothorn, T. 148
- Ieno, Elena N. 148
Ihaka, R. 3
- Kernighan, Brian W. 231
Kuh, E. 495
- Lafaye de Micheaux, P. 393, 437
Levene, H. 509
Liquet, B. 393
- Maindonald, J. 148
Mallows, C. L. 483
Matloff, N. 148
Meesters, Erik H.W.G. 148
Meyer, Carl 315
Miller, Jr. 511
Miller, K. W. 381
Montgomery, D. C. 515, 521
Muenchen, R. A. 4, 148
Murdoch, D. J. 148
- Neuwirth, Erich 4
- Park, S. K. 381
- Ritchie, D. M. 231
Robert, C. 382
Rupert, G. 511
- Sarkar, D. 148, 187

- Schwarz, G. [483](#)
Sleightholme, J. [231](#)
Stroustrup, B. [231](#)
Teetor, P. [148](#)
Vinod, H. D. [58](#)
- Weisberg, S. [455, 490](#)
Welsch, R. E. [495](#)
Wilks, A. R. [3](#)
Wilson, R. W. Jr. [483](#)
- Zuur, A. F. [148](#)

List of R Packages Mentioned in the Book

A

- ade4 253, 337, 373, 374
AnalyzeFMRI 191, 594
aplypack 365, 367, 368
asympTest 422, 430

B

- base 144
bigmemory 301
boot ... 405, 419, 421-423, 449, 577
Brobdingnag 132

C

- car ... 482, 507, 509, 516, 521, 533
caTools 169, 189
chron 115
combinat 232
ConvergenceConcepts 393

D

- datasets 144, 292
debug 257

E

- epitools 420, 421
evd 407

F

- ff 301
filehash 72
foreign 70, 71, 568, 581

G

- gdata ... 70, 93, 340, 440, 568, 583,
587, 600
ggplot2 187
gmodels 439, 519, 527
gputools 274
graphics 144

- grDevices 144
gregmisc 512, 513, 527
gtools 92, 93

I

- IndependenceTests 437

L

- lattice 6, 187
leaps 483
lmtest 464

M

- mapdata 191, 594
maps 191, 594
MASS 415
Matrix 316
misc3d 22
moments 352, 576

N

- nlme 523
nortest 448
numDeriv 327, 333

P

- parallel 274
plyr 96

R

- R.matlab 4, 71, 582
R2HTML 290, 532
rbenchmark 242
Rcmdr 7, 8, 515, 533, 535-538
RcmdrPlugin.sos 19
RcmdrPlugin.TeachingDemos .18
RColorBrewer .. 166, 167, 362, 363
RevoScaleR 301
RExcelInstaller 4

rgl	6, 187, 280, 415	T	
rgrs	353	tcltk	7, 8
Rmpi	273	tseries	463, 465
RODBC	76		
rsprng	274	U	
Ryacas	326	utils	144, 591
S		X	
snow	273, 274	xlsReadWrite	70, 73, 581
stats	144, 149	xtable	145

Solutions to Exercises

Solutions to Exercises from Chap. 3

- 3.1- [1] 1 2 3 4 5 6 7 8 9
- 3.2- [1] 2 4 6 8 10
- 3.3- The instruction `var<-3` does not output anything. The instruction `Var*2` outputs an error message because `Var` has not yet been defined.
- 3.4- The instruction `x<-2` does not output anything. The instruction `2x<-2*x` outputs an error message because a variable name should not begin with a number.
- 3.5- The instruction `root.of.four <- sqrt(4)` does not output anything. The instruction `root.of.four` outputs [1] 2.
- 3.6- The instruction `x<-1` does not output anything. The instruction `x< -1` outputs [1] FALSE.
- 3.7- The instruction `An even number <- 16` outputs an error message because a variable name should not contain spaces.
- 3.8- The instruction "An even number" <- 16 does not output anything.
- 3.9- The instruction "2x" <- 14 does not output anything.
- 3.10- The instruction `An even number` outputs an error message.
- 3.11- >2 +
+ 4
[1] 6
- 3.12- The instruction `TRUE + T + FALSE*F + T*FALSE + F` outputs [1] 2.
- 3.13- The five R data types are `numeric`, `complex`, `logical`, `character`, `raw`.
- 3.14- `X <- matrix(1:12,nrow=4,ncol=3,byrow=FALSE)`.
- 3.15- The R data structures are `c()`, `matrix()`, `array()`,
`list()`, `data.frame()`, `factor()`, `ordered()`.

Solutions to Exercises from Chap. 4

- 4.1-** The three main R functions to use to import data from an ASCII text file are `read.table()`, `scan()` and `read.ftable()`.
- 4.2-** `header`: a logical value indicating whether the file contains the names of the variables as its first line (e.g., `header=TRUE`).
`sep`: the field separator character. Values on each line of the file are separated by this character (e.g., `sep=" "` or `sep="\t"`).
`dec`: the character used in the file for decimal points (e.g., `dec=". "` or `dec=", "`).
`row.names`: a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names (e.g., `row.names=2`).
`skip`: the number of lines of the data file to skip before beginning to read data (e.g., `skip=4` to exclude the first 4 lines from reading).
`nrows`: the maximum number of rows to read in (e.g., `row.names=19`).
- 4.3-** Function `readLines()` reads some or all text lines from a connection.
- 4.4-** Function `fix()` enables one to modify a `data.frame` or a matrix using a small spreadsheet.
- 4.5-** `read.csv()`: reads a comma separated value file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file (note: `dec=". "`).
`read.csv2()`: reads a semicolon separated value file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file (note: `dec=", "`).
`read.delim()`: reads a tabulated separated value file (note: `dec=". "`).
`read.delim2()`: reads a tabulated separated value file (note: `dec=", "`).
- 4.6-** Function `read.ftable()` reads, writes and coerces flat contingency tables.
- 4.7-** Function `scan()` should be used when data are not organized in table format. Function `read.table()` is used for table format data sets.
- 4.8-** Importing data from an Excel sheet:
- Using copy–paste: select the data under Excel, copy these data to the clipboard and use the instruction:
- ```
x <- read.table(file("clipboard"), sep="\t", header=TRUE,
 dec=",")
```
- Using an intermediate ASCII file: save the Excel sheet as `.txt` (separator: TAB), then use function `read.table()`.
  - Using package `gdata` and function `read.xls()`.
- 4.9-** Package `foreign`.
- 4.10-** The `colClasses` argument from function `read.table()` can be used to indicate the type of each column and thus greatly increases the speed of reading of huge data sets.

**4.11-** Function `write.table()` enables one to write in a file the data set contained in a `data.frame`. Another function is `write()` that should be used for vector or matrix objects.

**4.12-** Here are four basic functions to build vectors:

- `c()`
- `seq()`
- `rep()`
- `":"()` (example `1:10`):

**4.13-** The instruction `seq(1, 2, by=0.1)` gives the following vector:

```
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

**4.14-** The instruction `rep(1:3, each=2)` gives the following vector:

```
1 1 2 2 3 3
```

**4.15-** The instruction `rep(1:3, 2)` gives the following vector:

```
1 2 3 1 2 3
```

**4.16-** Functions to enter data at hand in a small spread sheet are `data.entry()` and `de()`.

## Solutions to Exercises from Chap. 5

**5.1-** [1] 2 12

**5.2-** [,1] [,2]

```
[1,] 1 1
```

```
[2,] 2 2
```

**5.3-** Using functions `rownames()` and `colnames()`.

**5.4-** `cbind(X, Y)`

**5.5-** For the product of all elements of a matrix  $X$ : `prod(X)`.

For the product of the elements of each column of matrix  $X$ :

```
apply(X, FUN=prod, MARGIN=2).
```

**5.6-** [1] 4

```
[1] 2 6 8 3
```

**5.7-** `weight[height>180]`

**5.8-** [1] 7 8 9

```
[1] 5 6
```

**5.9-** `L[[4]] <- 1:10`

**5.10-** [1] 67

**5.11-** `attach(X)`

```
weight[sex=="F"]
```

```
height[sex=="F"]
```

```
or:
X[sex=="F",-3]
5.12- [1] 1 2 3 and integer(0)
5.13- [1] TRUE TRUE FALSE and [1] TRUE
5.14- [1] 4 4
5.15- [1] "acbd"
5.16- [[1]]
 [1] "ab" "cd"
5.17- [1] "" "cd"
5.18- tolower(c("Jack","Julia","William"))
5.19- strftime().
```

## Solutions to Exercises from Chap. 6

- 6.1-** `help(mean).`
- 6.2-** This instruction provides a list of all functions whose name contains the searched word.
- 6.3-** This instruction gives examples on using the searched function.
- 6.4-** Command `RSiteSearch()` enables one to search the website <http://search.r-project.org/nmz.html> directly from R. The extracted information is then displayed in your browser.
- 6.5-** 1. The header of the file, with:
- The name of the function for which we are looking for help.
  - The name of the package in which the function is included (e.g., `base`).
  - The origin of the help file: R Documentation.
2. An explicit title for the function (e.g., Arithmetic Mean).
3. A brief description of what the function does: Description.
4. How to use the function; in particular, the compulsory and optional arguments: Usage.
5. A description of the function's arguments: Arguments.
6. Explanations on the output of the function: Value.
7. References (statistical articles or books) related to the function's application domain: References.
8. The See Also section, which lists similar or related functions.
9. Examples of use: Examples.
- 6.6-** `help(package="stats")` or `library(help="stats")`.
- 6.7-** Type `data()` to see a list of data sets and then type the name of the chosen data set.

## Solutions to Exercises from Chap. 7

- 7.1- The `windows()` command is used to open a graphical device. The `dev.off()` command closes the window specified by *device-number* (if no device number is given, the active window is closed).
- 7.2- `savePlot(filename="myplot", type="pdf", device=dev.cur())`.
- 7.3- The instruction `par(mfrow=c(3, 2))` opens a graphical window where plots are successively displayed in a “matrix” with three rows and two columns (filled by rows).
- 7.4- Function `layout()` enables one to obtain a more evolved splitting of the graphical window than using function `par()`.
- 7.5- `points()`
- 7.6- `type="l"`
- 7.7- `abline()`
- 7.8- Function `curve()` enables one to draw any function of *x*.
- 7.9- The argument `col`.
- 7.10- Function `image()`. The instruction

```
image(as.matrix(rev(as.data.frame(t(X)))))
```

enables one to display coherently the image whose values are given in matrix *X*.

- 7.11- Function `text()`.
- 7.12- Function `identify()` or `locator()`.
- 7.13- The instruction `par(ask=TRUE)` outputs a message asking the user to press the ENTER key before each new plot is drawn.
- 7.14- `lty`
- 7.15- `pch`
- 7.16- `curve(cos(x), xlim=c(-10, 10), xlab="X axis", col="blue",  
main="Sinus and cosinus curves", ylim=c(-2,2),  
ylab="sin(x)")  
curve(sin(x), add=TRUE)  
abline(h=0, col="red")  
abline(v=0, col="red")  
arrows(3*pi/2, 1, pi/2, 1)  
text((3*pi)/2, 1, expression(hat(beta)[1]))`

## Solutions to Exercises from Chap. 8

- 8.1-**
- `function(name) {name}`: the class of the returned R object is `function`, and the display produced is `function(name) {name}`.
  - `(function(name) {name})("Ben")`: the class of the returned R object is `character`, and the display produced is "Ben"
  - `(function(name) {cat(name, "\n")})("Ben")`: the returned R object has no class, and the display produced is Ben
  - `(function(name) {invisible(name)})("Ben")`: the class of the returned R object is `character`, no display.

- 8.2-**
- No difference.
  - No difference.
  - No difference.

**8.3-** No difference when executing `name()` and `name("Peter")` if the function `name()` is defined by `name <- function(name="Peter") name` and also by `name <- function(name="Peter") name2 <- name`. For these two functions, the nature of `res`, obtained by `res <- name("Ben")` is of type `character`.

**8.4-** "Peter"

- 8.5-**
- "Ben L"
  - "Ben L"
  - "R D"

**8.6-** `name <- function(name="Peter") {cat(name, "\n")}`

**8.7-** For the execution of `names("peteR", "Ben", "R")`

- [1] "peteR" "Ben" "R"
- [[1]]  
[1] "peteR"

```
[[2]]
[1] "Ben"
```

```
[[3]]
[1] "R"
• [1] "peteR"
[1] "Ben"
[1] "R"
• [1] "peteR"
[1] "Ben"
[1] "R"
```

For the execution of `names(c("peteR", "L"), c("Ben", "L"), c("R", "D"))`

- [1] "peteR" "L" "Ben" "L" "R" "D"

- [[1]]
 

```
[1] "peteR" "L"
```
- [[2]]
 

```
[1] "Ben" "L"
```
- [[3]]
 

```
[1] "R" "D"
```
- [1] "peteR"
 

```
[1] "L"
[1] "Ben"
[1] "L"
[1] "R"
[1] "D"
```
- [1] "peteR" "L"
 

```
[1] "Ben" "L"
[1] "R" "D"
```

**8.8-** For the function `names <- function(names=c("Ben","R"),...)`  
`c(names,...)`

- [1] "PeteR"
- [1] "PeteR"
- [1] "PeteR"

For the function `names<-function(...,names=c("Ben","R"))`  
`c(names,...)`

- [1] "Ben" "R" "PeteR"
- ```
           name
"Ben"      "R" "PeteR"
```
- [1] "PeteR"

8.9- • `Male <- function(firstname,name) {`
 `obj <- list(firstname=firstname,name=name)`
 `class(obj) <- "Male"`
 `obj`
`}`

- `hello.Male <- function(obj)`
 `cat("Hello Mister",obj$firstname,obj$name,"!\n")`
- Hello Mister Ben L !
- Error : impossible to find function "hello"
- `hello <- function(obj) UseMethod("hello")`

8.10- • `Female <- function(firstname,name) {`
 `obj <- list(firstname=firstname,name=name)`
 `class(obj) <- "Female"`

- ```

 obj
}
• hello.Female <- function(obj)
 cat("Hello Mrs.",obj$firstname,obj$name,"!\n")
• Hello Mister Elsa R !
• Hello Mrs. Elsa R !
• Hello Mrs. Elsa R !
8.11- • Hello Mister Ben L !
 Hello Mrs. Elsa R !
• Error in UseMethod("hello") :
 no method for 'hello' applicable for
 an object of class "character"
• Hello Ben !
 Hello L !
 Hello Elsa !
 Hello R !

```

## Solutions to Exercises from Chap. 9

- 9.1-** `ls()` or `objects()`.
- 9.2-** `rm(foo)`.
- 9.3-** `getwd()`.
- 9.4-** `setwd(choose.dir())`.
- 9.5-** Save in a file (`name.RData`) the objects that have been created.
- 9.6-** The history of commands, the working environment, the contents displayed in the console and the graphs.
- 9.7-** This allows you to recall and replay previously typed commands (using arrows up and down on the keyboard).
- 9.8-** The function `history()` displays in a new window the list of all previous commands typed in the current session.
- 9.9-**
- ```

png(file="myplot.png")
curve(x**2)
dev.off()

```
- 9.10-** It enables one to access variables of a `data.frame` directly by their name.
- 9.11-** `require()` (or `library()`).
- 9.12-** Import a sequence of R instructions from a file to the console and check the syntax.

Solutions to Exercises from Chap. 10

10.1- `choose()`.

10.2- The instruction `sum(1:n)`.

10.3- `range()`.

10.4- The term by term product of the two following matrices:

$$\begin{bmatrix} [,1] & [,2] \\ [1,] & 1 & 0 \\ [2,] & 0 & 1 \end{bmatrix}$$

and

$$\begin{bmatrix} [,1] & [,2] \\ [1,] & 1 & 3 \\ [2,] & 2 & 4 \end{bmatrix}$$

which gives

$$\begin{bmatrix} [,1] & [,2] \\ [1,] & 1 & 0 \\ [2,] & 0 & 4 \end{bmatrix}$$

10.5- `%%%`.

10.6- Function `solve()` for the inverse and function `t()` for the transpose.

10.7- The instruction `diag(5)`.

10.8- Command `det()` for the determinant and `sum(diag())` for the trace.

10.9- `scale(A)`.

10.10- Function `eigen()`.

10.11- `myf <- function(x) {3*x^2+2}`
`integrate(myf,lower=-1,upper=1)`

10.12- `optimize(f=function(x)(sin(x))^2,lower=0,upper=2,`
`maximum=TRUE).`

10.13- Command `uniroot()` for a function and `polyroot()` for a polynomial.

Solutions to Exercises from Chap. 11

11.1- `table(x)/length(x)`.

11.2- `table(x,y)`.

11.3- `margin.table()`.

11.4- `prop.table()`.

11.5- `names(which.max(table(mytable)))`.

11.6- `diff(range(x))`.

11.7- `IQR(x)`.

11.8- `var(x)*(length(x)-1)/length(x)`.

11.9- `sqrt(var(x)*(length(x)-1)/length(x))/mean(x)`.

11.10- `mean(abs(x-mean(x)))`.

11.11- Package moments.

11.12- First, we need to compute the χ^2 statistic using:

```
chi2 <- summary(table(mytable))$statistic
```

Cramér's Φ^2 is obtained by $\chi2/N$.

11.13- Here is the code to compute the correlation ratio $\eta_{Y|X}^2$:

```
eta2 <- function(x, gp) {
  means <- tapply(x, gp, mean)
  frequency <- tapply(x, gp, length)
  varinter <- (sum(frequency * (means - mean(x))^2))
  vartot <- (var(x) * (length(x) - 1))
  res <- varinter/vartot
  return(res)
}
```

11.14- Function `barplot()` can be used to obtain a Pareto diagram.

11.15- A stacked bar chart can be obtained using the function `barplot()` with an object of type `matrix` as first argument.

11.16- Function `pie()` can be used to obtain a pie chart.

11.17- Function `boxplot()` can be used to obtain a box plot.

11.18- Function `hist()` is used to draw a histogram.

Solutions to Exercises from Chap. 12

12.1- `rnorm()`.

12.2- `rnorm(n, mean=2, sd=sqrt(10))` generates n realizations of a random variable with $\mathcal{N}(2, 10)$ distribution.

12.3- `qchisq()`.

12.4- `df()`.

12.5- `qt()`.

12.6- `pnorm(5, mean=4, sd=sqrt(2))-pnorm(3, mean=4, sd=sqrt(2))`.

12.7- `qnorm(0.95)`.

Solutions to Exercises from Chap. 13

- 13.1- `qbinom()`.
- 13.2- Function `pnorm()` is used to compute the Gaussian cumulative distribution function.
- 13.3- `t.test(x, conf.level=0.9)$conf.int`
- 13.4- Function `prop.test()` is used to compute a confidence interval for a proportion or to perform a statistical test for a proportion by approximating the binomial distribution by a normal distribution, whereas function `binom.test()` performs exact computations.
- 13.5- `ks.test()` and `wilcox.test()`.
- 13.6- `shapiro.test()`.
- 13.7- `chisq.test()` and `fisher.test()`.
- 13.8- Package `boot()`.
- 13.9- `paired=TRUE`.
- 13.10- The χ^2 test of independence enables one to test the dependence between two qualitative variables, whereas the χ^2 goodness-of-fit test can be used to test if a qualitative variable follows a given distribution. To perform these two tests, you can use the `chisq.test()` function. The χ^2 test of independence is performed on a `table`. For the χ^2 goodness-of-fit test, it is necessary to specify the `p` argument representing the theoretical probabilities of the possible outcomes of the qualitative variable.

Solutions to Exercises from Chap. 14

- 14.1- `lm(Y~X1)`.
- 14.2- `lm(Y~X1-1)` or `lm(Y~0+X1)`.
- 14.3- `lm(Y~X1+X2)`.
- 14.4- `lm(Y~X1+X2+X1:X2)` or `lm(Y~X1*X2)`.
- 14.5- `lm(Y~X1+I(X1^2)+I(X1^4))`.
- 14.6- `anova(lm(Y~X1+X2), lm(Y~X1+X2+X3+X4))`.
- 14.7- `residuals()`.
- 14.8- `coefficients()`.
- 14.9- The variable `Z` has to be introduced as a factor by means of the `factor()` command.
- 14.10- `lm(Y~poly(X, 3))`.
- 14.11- `add1()`.
- 14.12- `drop1()`.

Solutions to Exercises from Chap. 15

- 15.1-** `aov(Y~factor(A))`.
- 15.2-** `aov(Y~factor(A)+factor(B))`.
- 15.3-** `aov(Y~factor(A)*factor(B))`.
- 15.4-** `bartlett.test()` and `levene.test()`.
- 15.5-** `pairwise.t.test()`.
- 15.6-** `summary(lm(Y~factor(X)))`.
- 15.7-** The function `C()`.

Solutions to Worksheet

Solutions to Worksheet from Chap. 3

Study of Body Mass Index

```
3.1- Individuals <- c("Edward", "Cynthia", "Eugene", "Elizabeth",
  "Patrick", "John", "Albert", "Lawrence",
  "Joseph", "Leo")
Weight <- c(16, 14, 13.5, 15.4, 16.5, 16, 17, 14.8, 17, 16.7)
Height <- c(100, 97, 95.5, 101, 100, 98.5, 103, 98, 101.5, 100)
Gender <- c("F", "F", "M", "F", "M", "M", "M", "M", "M", "M")

3.2- mean(Weight) # result : [1] 15.69
mean(Height) # result : [1] 99.45

3.3- BMI <- Weight/(Height/100)^2
BMI      # Outputs the results

3.4- myTable <- data.frame(Individuals, Weight, Height, Gender, BMI)
myTable    # Outputs the results

3.5- help(plot)

3.6- plot(Height, Weight, main="Scatter plot of Weight as a
function of Height")
```

Solutions to Worksheet from Chap. 4

Reading Various Data Sets

A- Entering Data from a Hard Copy

- Cold sore

4.1- `blisters <- as.data.frame(de(""))`

First, change the type of each column to Real and the names to trt_i , $1 \leq i \leq 5$ (by clicking on the first cell of each column), then enter your data. Click on Quit at the end. Next, we can type the following command to display the data in R:

`blisters # Outputs the data.`

4.2- `attach(blisters)`

```

mean(trt1) # Result: [1] 7.5
mean(trt2) # Result: [1] 5
mean(trt3) # Result: [1] 4.333333
mean(trt4) # Result: [1] 5.166667
mean(trt5) # Result: [1] 6.166667
```

4.3- We obtain directly the same results by typing

`colMeans(blisters).`

4.4- `write.table(file="blisters.txt",blisters,row.names=FALSE).`

4.5- You can use your favourite text editor to display the contents of the file `blisters.txt` and thus check if it has been correctly created (the instruction `getwd()` will tell you where this file has been saved).

4.6- `ls()`

```

rm(blisters)
ls()
blisters # We see that the object blisters has disappeared.
```

4.7- `blisters <- read.table("blisters.txt",header=TRUE,sep="")`
`blisters # Here is again!`

- Risk factors for atherosclerosis

4.1- Enter data as they are organized in the contingency table, by pressing the ENTER key after each end of line. After the last line, press once more on the ENTER key.

```

X <- scan() # Data are then collected in a vector.
X <- matrix(X,ncol=3,nrow=6,byrow=TRUE)

4.2- class(X) <- "ftable"

4.3- attributes(X)$col.vars<-list(alcohol=c("nondrinker",
                                             "occasional-drinker","regular-drinker"))
       attributes(X)$row.vars<-list(GENDER=c("M","F"),tobacco=
                                         c("non-smoker","former smoker","smoker"))

4.4- X

4.5- write.ftable(X,file="athero.txt").

4.6- You can use your favourite text editor to display the contents of the file
      athero.txt and thus check that it has been correctly created (the instruction
      getwd() will tell you where this file has been saved).

4.7- rm(X)
      X # The object has been removed.

4.8- X <- read.ftable(file="athero.txt")
      X # Here it is again!

```

B- Importing from Other Software

4.1- You can begin by downloading the file "<http://biostatisticien.eu/springeR/bmichild.xls>" using your browser. If you have Excel, you can then transform it into the file bmichild.txt, with TAB as separators, then use the following R command:

```
bmi.XLS <- read.table(file.choose(),header=TRUE,sep="\t",
                      dec=",")
```

If you do not have Excel, you can use the package required `xlsReadWrite` after installing it:

```
require("xlsReadWrite")
bmi.XLS <- read.xls("bmichild.xls")
```

Finally, under Linux, you can use the commands:

```
require("gdata")
bmi.XLS <- read.xls("http://biostatisticien.eu/springeR/
                      bmichild.xls")
```

4.2- Install the package `foreign`. Download the file "<http://www.biostatisticien.eu/springeR/bmichild.xpt>" using your browser.

```
require("foreign")
bmi.SAS <- read.xport(file.choose()) # Select bmichild.xpt.
```

4.3- url <- "http://www.biostatisticien.eu/springeR/
bmichild.sav"
bmi.SPSS <- read.spss(url)
bmi.SPSS <- as.data.frame(bmi.SPSS)

4.4- Install the package R.matlab.

```
require("R.matlab")
x <- readMat("http://www.biostatisticien.eu/springeR/
              bmichild.mat")
class(x) # x is a list.
x       # We see that the data are in $bmi[,,1].
x <- x$bmi[,,1]
# Note that the elements of GENDER and zep are saved into
# a list.
x$GENDER
class(x$GENDER) <- "character"
x$GENDER
class(x$zep) <- "character"
bmi.MAT <- as.data.frame(x)
```

4.5- summary(bmi.XLS)
summary(bmi.SAS)
summary(bmi.SPSS)
summary(bmi.MAT)

4.6- write.table(bmi.SPSS,"bmichild.txt",row.names=FALSE)

C- Importing More Complex Data Files

4.1- readLines("http://biostatisticien.eu/springeR/
 geoidformat.txt")
X <- scan("http://biostatisticien.eu/springeR/raf98.gra",
 skip=3)
X <- matrix(X,ncol=421,nrow=381,byrow=TRUE)
dim(X)

4.2- Save the file <http://biostatisticien.eu/springeR/Infarction.xls> using your browser, then transform it into a file Infarction.txt (TAB as separator). Next, use the command:

```
infarction <- read.table("Infarction.txt",header=TRUE,
                         sep="\t",na.strings = ".",dec=",")
```

Under Linux, rather use

```
require("gdata")
url <- "http://biostatisticien.eu/springeR/Infarction.xls"
```

```
infarction <- read.xls(url,header=TRUE,sep=",",
na.strings=". ",dec=". ")
```

- 4.3-** Download the file "http://biostatisticien.eu/springeR/nutrition_elderly.txt" in the folder whose name is given by the command `getwd()`, then enter the following commands:

```
X <- read.table("nutrition_elderly.txt",row.names=1)
X <- t((X))
X <- as.data.frame(X)
```

- 4.4-** `url <- "http://www.biostatisticien.eu/springeR/ Birth_weight.txt"`
- `readLines(url) # To get an idea of the file organization.`
`# Note the three missing lines [33] to [35],`
`# and also the presence of dashes at line [194].`
- ```
X <- read.table(url,row.names=1,skip=1,header=FALSE,
sep=";",nrows=189, blank.lines.skip=TRUE)
Y <- read.table(url,nrows=1,row.names=1)
colnames(X) <- as.matrix(Y)
head(X) # Display the first lines of X.
```

## Solutions to Worksheet from Chap. 5

### Manipulating Various Data Sets

#### A- Manipulating a few data sets presented at the beginning of the book

File nutrielderly.xls

- 5.1-** Install the package `gdata`. Be careful that the use of this package necessitates the PERL software which is not available by default under Windows. Thus install also PERL (<http://www.biostatisticien.eu/springeR/Rtools.exe>):

```
require("gdata")
nutrien1 <- read.xls("http://www.biostatisticien.eu/
springeR/nutrien1.xls")
nutrien2 <- read.xls("http://www.biostatisticien.eu/
springeR/nutrien2.xls")
colnames(nutrien1) <- tolower(colnames(nutrien1))
result <- rbind(nutrien1,nutrien2)
```

```

5.2- nutrien3 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien3.xls")
nutrien4 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien4.xls")
result <- merge(nutrien3,nutrien4,all=TRUE)

5.3- nutrien5 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien5.xls")
nutrien6 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien6.xls")

```

The coding of the variables is given in the table in Sect. 2.4. We will try to identify abnormal values using the following instruction:

```
apply(nutrien5,FUN=table,MARGIN=2)
```

We can identify six abnormal values in `nutrien5`: `gender=12`, `gender=21`, `height=1.67`, `weight=200`, `age=8` and `chocol=7`. The subjects with these abnormal values are identified using

```

ind5gender <- nutrien5$Subject[which(nutrien5$gender %in%
 c(12,21))]
ind5height <- nutrien5$Subject[which(nutrien5$height ==
 1.67)]
ind5weight <- nutrien5$Subject[which(nutrien5$weight==200)]
ind5age <- nutrien5$Subject[which(nutrien5$age==8)]
ind5chocol <- nutrien5$Subject[which(nutrien5$chocol==7)]
ind5abnorm <- c(ind5gender,ind5height,ind5weight,ind5age,
 ind5chocol)

```

Let us do the same with `nutrien6`.

```
apply(nutrien6,FUN=table,MARGIN=2)
```

We find seven abnormal values in `nutrien6`: `gender=12`, `gender=21`, `weight=8`, `age=7`, `meat=6`, `chocol=6` and `fat=9`. The subjects with these abnormal values are identified using

```

ind6gender <- nutrien6$Subject[which(nutrien6$gender %in%
 c(12,21))]
ind6weight <- nutrien6$Subject[which(nutrien6$weight==8)]
ind6age <- nutrien6$Subject[which(nutrien6$age==7)]
ind6meat <- nutrien6$Subject[which(nutrien6$meat==6)]
ind6chocol <- nutrien6$Subject[which(nutrien6$chocol==6)]
ind6fat <- nutrien6$Subject[which(nutrien6$fat==9)]
ind6abnorm <- c(ind6gender,ind6weight,ind6age,ind6meat,
 ind6chocol,ind6fat)

```

We will now look at whether it is possible to correct some of these abnormal values. To do this, we will investigate whether an individual with an abnormal

value in one of the tables may be present in another table with a normal value. Here is the list of common subjects between both files:

```
ind.com <- intersect(nutrien6$Subject, nutrien5$Subject)
```

List of abnormal subjects in nutrien5 that are also present in nutrien6:

```
intersect(ind.com, ind5abnorm)
Result : no subject.
```

List of abnormal subjects in nutrien6 that are also present in nutrien5:

```
intersect(ind.com, ind6abnorm)
Result: subject 30
nutrien5[nutrien5$Subject==30,]
nutrien6[nutrien6$Subject==30,]
We have gender=21 in nutrien6 and gender=2 in nutrien5.
```

We thus replace this value in nutrien6:

```
nutrien6$gender[which(nutrien6$Subject==30)] <-
 nutrien5$gender[which(nutrien5$Subject==30)]
```

We can now merge the two tables:

```
result <- merge(nutrien5, nutrien6, all=TRUE)
```

Without other information on abnormal data, we decide not to change them for the moment.

**5.4-**

```
nutrien7 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien7.xls")
nutrien8 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien8.xls")
result <- cbind(nutrien7, nutrien8)
```

**5.5-**

```
nutrien9 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien9.xls")
nutrien10 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien10.xls")
apply(nutrien9, FUN=table, MARGIN=2, useNA="ifany")
apply(nutrien10, FUN=table, MARGIN=2, useNA="ifany")
```

The variable chocol contains 29 missing data points. We delete this variable before combining the two tables:

```
nutrien9bis <- nutrien9[,-which(colnames(nutrien9)==
 "chocol")]
result <- cbind(nutrien9bis, nutrien10)
```

**5.6-**

```
nutrien11 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien11.xls")
```

```

nutrien12 <- read.xls("http://www.biostatisticien.eu/
 springeR/nutrien12.xls")
Number of missing values for all individuals in each
table:
ind11NA <- apply(is.na(nutrien11),FUN=sum,MARGIN=1)
ind12NA <- apply(is.na(nutrien12),FUN=sum,MARGIN=1)
Determination of the individual with the most missing
values:
ind11max <- which.max(ind11NA) # We find the individual 86.
ind12max <- which.max(ind12NA) # We find the individual 86.
ind11NA[ind11max] # 3 missing values.
ind12NA[ind12max] # 4 missing values.

```

The individual 86 has 7 missing values on 13 variables. We suppress it from the merged table:

```
result <- cbind(nutrien11,nutrien12)[-86,]
```

```
5.7- url <- "http://www.biostatisticien.eu/springeR/
 nutri_elderly.xls"
nutri_elderly <- read.xls(url)
nrow(nutri_elderly[nutri_elderly$fish==0 & nutri_elderly
$meat==0,])
```

There is no vegetarian among the individuals studied.

#### File Intima\_Media\_Thickness.xls

```
5.1- url <- "http://www.biostatisticien.eu/springeR/
 Intima_Media_Thickness.xls"
Intima <- read.xls()
Intima <- transform(Intima,BMI=weight/(height/100)^2)
or equivalently:
BMI <- Intima$weight/((Intima$height/100)^2)
Intima <- cbind(Intima,BMI)

5.2- Intima$measure[Intima$BMI>30]

5.3- Intima[Intima$SPORT==1,]

5.4- Intima[Intima$BMI<=30 & Intima$AGE>=50,]
```

## File bmichild.xls

```

5.1- bmichild <- read.xls("http://www.biostatisticien.eu/
 springeR/bmichild.xls")
bmichild <- transform(bmichild,BMI=weight/(height/100)^2)
or equivalently:
BMI <- bmichild$weight/((bmichild$height/100)^2)
bmichild <- cbind(bmichild,BMI)

5.2- subset(bmichild,BMI<15 & an<=3.5 & month <=5)
or equivalently:
bmichild[bmichild$BMI<15 & bmichild$an <=3 &
bmichild$month<=5,]

5.3- sum(bmichild$BMI<15 & bmichild$an<=3 & bmichild$month<=5)
or equivalently:
nrow(bmichild[bmichild$BMI<15 & bmichild$an<=3 &
bmichild$month<=5,])

```

We find seven children.

## File Birth\_weight.xls

```

5.1- url <- "http://www.biostatisticien.eu/springeR/
 Birth_weight.xls"
birth.weight <- read.xls(url)
PTL1 <- birth.weight$PTL
PTL1[birth.weight$PTL>=2] <- 2
birth.weight <- cbind(birth.weight,PTL1)

5.2- FVT1 <- birth.weight$FVT
FVT1[birth.weight$FVT>=2] <- 2
birth.weight <- cbind(birth.weight,FVT1)

5.3- birth.weight[order(birth.weight$BWT),]

5.4- birth.weight[birth.weight$RACE<=2 & birth.weight$SMOKE==1,]

```

**B- Handling Missing Values**

Install the package gdata. And since this package necessitates PERL, also install PERL (<http://www.biostatisticien.eu/springeR/Rtools.exe>). Now, we read the data:

```
url <- "http://www.biostatisticien.eu/springeR/Infarction.xls"
read.xls(url)
```

We note that the missing values are coded with ". ". We thus use the following instruction:

```
infarction <- read.xls(url,na.strings=". ")

5.1- indNA <- which(apply(is.na(infarction),FUN=any,MARGIN=1)
 ==TRUE)

5.2- sup1 <- function(x) {sum(x)>1}
 indNA <- which(apply(is.na(infarction),FUN=sup1,
 MARGIN=1)==TRUE)
 infarction$NUMBER[indNA]

5.3- colnames(infarction)[which(apply(is.na(infarction),FUN=any,
 MARGIN=2)==TRUE)]

5.4- (a) infarction[as.logical(apply(!is.na(infarction),1,
 prod)),]
 (b) infarction[!apply(apply(infarction,1,is.na),2,any),]
 (c) infarction[apply(!apply(infarction,1,is.na),2,all),]
 (d) infarction[complete.cases(infarction),]
 (e) na.omit(infarction)
```

## C- Handling Character Strings

```
5.1- url<-"http://www.biostatisticien.eu/springeR/dept-pop.csv"
 dept <- read.csv(url,dec=",")

5.2- numdep <- substring(dept$Departement,1,3)
 Dept <- substring(dept$Departement,4)
 dept <- cbind(numdep,Dept,dept[, -1])
```

## D- Influenza Epidemics in France Since 1984

```
5.1- url <- "http://www.biostatisticien.eu/springeR/flu.csv"
 head(read.csv(url))
```

We note that the missing values are coded using dashes (-). We thus use the following instruction:

```
flu <- read.csv(url,na.strings="-")
```

```
5.2- names(flu)
 flu$date
```

```
5.3- unique(sort(substring(flu$date,5,6)))
```

We find the numbers "01" to "53".

**5.4-** `strftime("198444",format="%Y%W")`

We notice that the above instruction gives the correct year, but in fact the current day and month.

**5.5-** We read on the calendar that the first Monday of the 44th week corresponds to October 29, 1984.

**5.6-** `strftime("1984441",format="%Y%W%w")`

**5.7-** `flu$Date[1:10]`

```
strftime(paste(as.character(flu$Date[1:10]),"1",sep=""),
 format="%Y%W%w")
```

We notice on the website calendar that this does not work for December 31, 1984. We should rather use

```
strftime("1984531",format="%Y%W%w")
```

whereas we used

```
strftime("1985011",format="%Y%W%w")
```

Thus there is a problem with week format.

**5.8-** `date1 <- as.POSIXlt("29,10,1984",format="%d,%m,%Y")`

**5.9-** `date1`

```
date1+7
```

The above operation added 7 s to `date1`.

**5.10-** `date1+7*24*60*60`

**5.11-** `dates <- date1 + 7*24*60*60*(0:(nrow(flu)-1))`

**5.12-** `flu$Date <- substring(dates,1,10)`

```
5.13- mask1 <- (as.POSIXlt(dates) >= as.POSIXlt("1992-09-15"))
 mask2 <- (as.POSIXlt(dates) <= as.POSIXlt("1993-11-03"))
 portion <- flu[mask1 & mask2,]
```

**5.14-** `flu.cases <- colSums(portion[, -1],na.rm=TRUE)`  
`flu.cases`

## E- Combining Tables or Lists and Other Manipulations

**5.1-** `a <- matrix(1:6,3,2)`  
`rownames(a) <- c(1,2,6)`  
`b <- matrix(1:8,4,2)`  
`rownames(b) <- c(3,4,5,7)`

```

5.2- ab <- rbind(a,b)
 ab <- ab[order(rownames(ab)),]
 ab

5.3- list1 <- list()
 list1[[1]] <- matrix(runif(25),nr=5)
 list1[[2]] <- matrix(runif(30),nr=5)
 list1[[3]] <- matrix(runif(15),nr=5)

 matrix(unlist(list1),nrow=5)
 # or also:
 do.call(cbind,list1)

5.4- list2 <- list()
 list2[[1]] <- matrix(runif(25),nc=5)
 list2[[2]] <- matrix(runif(35),nc=5)
 list2[[3]] <- matrix(runif(15),nc=5)

 tmp <- lapply(list2,FUN=t)
 t(matrix(unlist(tmp),nrow=5))
 # or also:
 do.call(rbind,list2)

5.5- tmp <- data.frame(Disease = c("Infarction", "Hepatitis",
 "Lung cancer"),RF = c("tobacco, alcohol", "alcohol",
 "tobacco"))
 tmp
 tmp$Disease[grep("tobacco",tmp$RF)]

```

## F- The French Chevalier de Méré

- fourthrows <- function() {
 asixormore <- 0
 game <- sample(1:6,4,replace=TRUE)
 nbsix <- sum(game == 6)
 if(nbsix >= 1) asixormore <- 1
 return(asixormore)
 }
- twentyfourthrows <- function() {
 adoublesixormore <- 0
 die1 <- sample(1:6,24,replace=TRUE)
 die2 <- sample(1:6,24,replace=TRUE)
 nbdoublesix <- sum(die1[which(die1 == die2)] == 6)
 if(nbdoublesix >= 1) adoublesixormore <- 1
 return(adoublesixormore)
 }

```

• meresix <- function(nsim = 2000) {
 atleastasix <- 0
 atleastadoublesix <- 0
 for (j in 1:nsim) {
 atleastasix <- atleastasix + fourthrows()
 atleastadoublesix <- atleastadoublesix +
 twentyfourthrows()
 }
 ***** Displaying the output *****
 cat("Frequency of 6s =",atleastasix/nsim,"\n")
 cat("Frequency of double-6s =",atleastadoublesix/nsim,
 "\n")
}

```

## Solutions to Worksheet from Chap. 6

### Where to Find Information

**6.1-** We type

```
help.search("combination")
```

then we note the right function. We choose the one from package `utils` which is present by default in R.

```
help(combn)
```

**6.2-** `combn(c(5,8,2,9),3)`

**6.3-** `help.search("crime")`  
`help(USArrests)`

**6.4-** `dim(USArrests)` # Dimension of the data set.  
`names(USArrests)` # Names of variables.  
`rownames(USArrests)` # Names of individuals (states).

- 6.5-** Subscribe to <https://stat.ethz.ch/mailman/listinfo/r-help>.
- 6.6-** Read the General Instructions on the R Mailing Lists page and the posting guide on <http://www.r-project.org/posting-guide.html>.
- 6.7-** To unsubscribe, it suffices to follow the instructions given at the bottom of page <https://stat.ethz.ch/mailman/listinfo/r-help>.
- 6.8-** Connect on IRC, for example, via the website <http://webchat.freenode.net>.

- 6.9-** Subscribe to <http://stackoverflow.com/questions/tagged/r>.
- 6.10-** Read the Windows FAQ from <http://cran.r-project.org/faqs.html>.  
Section 7.5 explains TAB completion mechanism.
- 6.11-** Enter a quote ("), then press twice on the TAB key.

## Solutions to Worksheet from Chap. 7

### Creating Various Plots

#### A- Complex Numbers

```
7.1- z <- 1+2i
plot(1,3,xlab="Re(z)",ylab="Im(z)",xlim=c(0,2.5),
 ylim=c(0,2.5),
 main="Complex numbers")
segments(0,0,Re(z),Im(z))
points(Re(z),Im(z),pch=19)
abline(h=0,v=0)
segments(Re(z),0,Re(z),Im(z),lty=3)
segments(0,Im(z),Re(z),Im(z),lty=3)
text(Re(z),Im(z),"z",pos=4)
text(0.4,1.1,"Mod(z)",srt=Arg(z)*180/pi)
r<-0.5
x<-seq(from=Re(r*exp(1i*Arg(z))),to=r,length=100)
y<-sqrt(0.5^2-x^2)
points(x,y,type="l")
text(0.55,0.35,"Arg(z)",srt=-45,cex=0.8)
```

#### B- Flag of Canada

```
7.1- require("caTools")
7.2- Save the file "http://www.biostatisticien.eu/springeR/canada.gif" in your working environment. Then launch the instruction:
X <- read.gif("canada.gif")
7.3- image(as.matrix(rev(as.data.frame(t(X$image)))),col=X$col)
```

```

7.4- coord <- locator(25) # Retrieve coordinates of points
 # constitutive of the contour maple
 # leaf.
rec1 <- locator(2) # Retrieve coordinates of bottom-left
 # and top-right vertices of left
 # rectangle.
rec2 <- locator(2) # Retrieve coordinates of bottom-left
 # and top-right vertices of right
 # rectangle.
windows() # or X11() under Linux.
plot(0,xlim=c(0,1),ylim=c(0,1),type="n",ann=FALSE,
 axes=FALSE)
rect(rec1$x[1],rec1$y[1],rec1$x[2],rec1$y[2],col="red")
rect(rec2$x[1],rec2$y[1],rec2$x[2],rec2$y[2],col="red")
polygon(coord,col="red")

```

## C- Frequency Tables

**7.1-** Enter the data set:

```

X <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1,1,
 1,
 1,
 3,2,1,1,1,1,1,1,2,1,4,1,1,1,1,1,2,2,1,1,1,1,2,3,3,1,2,1,1,
 1,1,1,3,4,1,1,1,1,2,3,4,3,1,2,1,1,4,1,1,4,4,1,1),ncol=8)
colnames(X) <- c("Nr","W1","W2","W3","W4","W5","W6","W7")
fi <- tabulate(X[, "W7"], 4)/16
as.vector(t(cbind(fi, 1-fi)))

```

**7.2-** For this, create a function:

```

f <- function(x){
 fi <- tabulate(x,4)/16
 as.vector(t(cbind(fi,1-fi)))
}
freq <- apply(X[,-1],FUN=f,MARGIN=2)
rownames(freq) <- c("f1","1-f1","f2","1-f2","f3","1-f3",
 "f4","1-f4")

```

**7.3-** barplot(freq,col=c("black","white"))

**7.4-** windows() # or X11() under Linux.

```

barplot(freq,col=c("red","white"),width=1,space=0.1,axes=F,
 border="black",names.arg=rep("",7))
axis(2,labels=1:4,at=1:4-0.5,lty=0,las=1,col.axis="blue")
par(cex=.8)

```

```

axis(3,labels=c("W1","W2","W3","W4","W5","W6","W7"),
 at=0:6+0:6/10+0.5,lty=0,col.axis="blue")
par(cex=1)
title(main="Burning scores",col="black")

```

## D- Anatomic Images of the Brain

- 7.1- Read the data:

```

dim <- 256
url <- "http://www.biostatisticien.eu/springeR/anat.img"
bytes <- readBin(url,what="raw",n=dim*dim*2)

```

- 7.2- Permutate pairs of bytes:

```

nbval <- dim^2
indices <- rbind((1:nbval)*2,(1:nbval)*2-1)
indices <- as.numeric(indices)
x <- bytes[indices]

```

- 7.3- Transformation to decimal representation:

```

x <- matrix(x,ncol=2,byrow=TRUE)
test <- function(x) {
 as.numeric(paste("0x",paste(x,collapse=""),sep=""))
}
values <- apply(x,MARGIN=1,FUN=test)

```

- 7.4- X <- matrix(values,nrow=dim,ncol=dim)

- 7.5- image(X,col=gray(0:100 / 100))

- 7.6- Install the package AnalyzeFMRI. Download the files anat.img and anat.hdr to your workspace, then type the following instructions:

```

require("AnalyzeFMRI")
Y <- f.read.volume("anat.img")
image(Y[, , 1],col=gray(0:1000 / 1000))

```

## E- Drawing the Map of a Region of France

- 7.1- Install the *packages* maps and mapdata, then load them into memory:

```

require("maps")
require("mapdata")

```

- 7.2- map("france")

- 7.3- france <- map("france",plot=FALSE)

- 7.4-** Latitude/longitude data are organized in `france$x/france$y` for each region in `france$names` (until the next NA).

- 7.5-** Vector of NA indices:

```
indNA <- which(is.na(france$x))
```

- 7.6-** deptname <- "Gard"

- 7.7-** inddept <- which(france\$names==deptname)

- 7.8-** plot(france\$x[indNA[inddept-1]:indNA[inddept]],  
      france\$y[indNA[inddept-1]:indNA[inddept]],  
      type="l",main=deptname,xlab="Longitude",  
      ylab="Latitude")

- 7.9-** Retrieve coordinates of the town called Alès on <http://www.gpsvisualizer.com/geocode>.

```
ales <- c(4.08268,44.121288)

points(ales[1],ales[2],pch=16,col="red")

text(ales[1],ales[2],"Al\`es",pos=1)
```

## F- Representation of the Geoid in France

- 7.1-** Read the data:

```
readLines("http://www.biostatisticien.eu/springeR/

 geoidformat.txt")

data <- scan("http://www.biostatisticien.eu/springeR/

 raf98.gra",skip=3)
```

- 7.2-**
- ```
Z <- matrix(data,nrow=421,ncol=381,byrow=FALSE)  

Z <- as.matrix(rev(as.data.frame(Z)))  

layout(mat=matrix(1:2,ncol=2),widths=c(5,1))  

par(mar=c(4,4,3,0),mai=c(1, 1, 1, 0),las=1,tck=-0.01)  

x<-seq(from=-5.5,to=8.5,length=421)  

y<-seq(from=42,to=51.5,length=381)  

image(x,y,Z,col=rainbow(17),xlab="Longitude",  

      ylab="Latitude",axes=FALSE)  

par(ps=18)  

title("The quasigeoid QGF98 model",family="HersheyScript")  

par(ps=11)  

axis(1,at=(-6):8,labels=paste((-6):8,"o",sep=""))  

axis(2,at=42:51,labels=paste(42:51,"o",sep=""))  

contour(x,y,Z,add=TRUE)  

par(mai=c(0, 0, 0, 0),bty="n")
```

```
plot(0:1,0:1,axes=FALSE,xlab="",ylab="",type="n")
legend("center",legend=42:55,fill=rainbow(17),bty="n",
      title="Meters")
```

Solutions to Worksheet from Chap. 8

Programming Functions and Object-Oriented Programming in R

A- Managing a Bank Account

```
8.1- .folder.accounts <- "./Account"
path.account <- function(name) {
  file.path(.folder.accounts,paste(name,".RData",sep=""))
}

8.2- account <- function(name) {
  path <- path.account(name)
  account <- data.frame(amount=numeric(0),
                         mode=factor(levels=c("Debit","Credit")),
                         date=character(0),remark=character(0))
  save(account,file=path)
  cat("Account",name,"created !\n")
}

8.3- debit <- function(name,amount,remark="",
                      date=format(Sys.time(),"%d/%m/%Y")) {
  path <- path.account(name)
  load(path)
  account <- rbind(account,data.frame(amount=amount,
                                         mode="Debit",date=date,remark=remark))
  save(account,file=path)
}

credit <- function(name,amount,remark="",
                     date=format(Sys.time(),"%d/%m/%Y")) {
  path <- path.account(name)
  load(path)
  account <- rbind(account,data.frame(amount=amount,
                                         mode="Credit",date=date,remark=remark))
  save(account,file=path)
}
```

8.4- The instruction

```
sum(account[account$mode=="Credit", "amount"])
```

returns the credited amount on the account. We modify the function `account()` this way:

```
account <- function(name) {
  path <- path.account(name)
  if(!file.exists(path)) {
    account <- data.frame(amount=numeric(0), mode=
      factor(levels=c("Debit", "Credit")),
      date=character(0),
      remark=character(0))
    save(account, file=path)
    cat("Account", name, "created !\n")
  } else {
    load(path)
    cat("State of account", name, "=",
        sum(account[account$mode=="Credit", "amount"])
        - sum(account[account$mode=="Debit", "amount"]),
        "euros.\n")
  }
}
```

B- Organizing Graphical Objects

8.1- `Window <- function(x=0,y=0,width=2,height=2,log="") {
 obj <- list(x=x,y=y,width=width,height=height,log=log)
 class(obj) <- "Window"
 obj
}`

8.2- `Circle <- function(x=0,y=0,radius=0.5) {
 circle <- list(x=x,y=y,radius=radius)
 class(circle) <- "Circle"
 circle
}`

```
Rectangle <- function(x=0,y=0,width=1,height=height) {
  rectangle <- list(x=x,y=y,width=width,height=height)
  class(rectangle) <- "Rectangle"
  rectangle
}
```

```
8.3- plot.Window <- function(obj) {  
  plot.new()  
  plot.window(xlim=obj$x+c(-1,1)*obj$width/2,  
              ylim=obj$y+c(-1,1)*obj$height/2,obj$log,asp=1)  
}  
  
plot.Rectangle <- function(rectangle) {  
  rect(rectangle$x-rectangle$width/2,rectangle$y-  
       rectangle$height/2,rectangle$x+rectangle$width/2,  
       rectangle$y+rectangle$height/2)  
}  
  
plot.Circle <- function(circle) {  
  symbols(circle$x,circle$y,circle=circle$radius,  
          inches=FALSE,add=TRUE)  
}  
  
8.4- mywindow <- Window(0,0,2,2)  
mycircle <- Circle(0,0,.5)  
myrectangle <- Rectangle(0,0,1,1)  
plot(mywindow);plot(mycircle);plot(myrectangle)  
  
8.5- MyPlot <-function(x=0,y=0,width=2,height=2,log="") {  
  graph <- list(objects=list())  
  class(graph) <- "MyPlot"  
  graph  
}  
  
8.6- add.MyPlot <- function(graph,...) {  
  graph$objects <- c(graph$objects,list(...))  
  graph  
}  
  
add <- function(obj,...) UseMethod("add")  
  
plot.MyPlot <- function(graph) {  
  for(object in graph$objects) {  
    plot(object)  
  }  
}  
  
graph <- MyPlot()  
graph <- add(graph,Window(0,0,2,2),Circle(0,0,.5),  
            Rectangle(0,0,1,1))  
plot(graph)
```

8.7- MyPlot <- function(...,x=0,y=0,width=2,height=2,log="") {
graph <- list(objets=list())
class(graph) <- "MyPlot"
graph <- add(graph,Window(x,y,width,height,log),...)
graph
}

graph <- MyPlot(Circle(),Rectangle())
plot(graph)

8.8- display <- function(obj,...) UseMethod("plot")
graph <- MyPlot(Circle(),Rectangle())
display(graph)

8.9- Your turn to play!

C- Creating a Class lm2 for Linear Regression with Two Regressors

```
lm2 <- function(...) {
  obj <- lm(...)
  if(ncol(model.frame(obj))!=3)
    stop("two independent variables are required!")
  class(obj) <- c("lm2",class(obj)) # or c("lm2","lm")
  obj
}

n <- 20
x1 <- runif(n,-5,5)
x2 <- runif(n,-50,50)
y <- .3+2*x1+2*x2+rnorm(n,0,20)
reg2 <- lm2(y~x1+x2)
summary(reg2)

plot3d.lm2 <- function(obj, radius=1, lines=TRUE, windowRect, ... ) {
  matreg <- model.frame(obj)
  colnames(matreg) <- c("y", "x1", "x2")
  predlim <- cbind(c(range(matreg[,2])), rev(range(matreg[,2]))),
  rep(range(matreg[,3]),c(2,2)))
  predlim <- cbind(predlim,apply(predlim,1,
    function(l) sum(c(1,1)*coef(obj))
  ))
  if(missing(windowRect)) windowRect=c(2,2,500,500)
  open3d(windowRect=windowRect,...)
  bg3d(color = "gray")
  plot3d(formula(obj),type="n")
```

```

spheres3d(formula(obj),radius=radius,specular="green")
quads3d(predlim,color="blue",alpha=0.7,shininess=128)
quads3d(predlim,color="cyan",size=5,front="lines",
          back="lines",lit=F)
if(lines) {
  matpred <- cbind(matreg[2:3],model.matrix(obj)%%coef(obj))
  points3d(matpred)
  colnames(matpred) <- c("x1","x2","y")
  matlines <- rbind(matreg[,c(2:3,1)],matpred)
  nr <- nrow(matreg)
  matlines <- matlines[rep(1:nr,rep(2,nr))+c(0,nr),]
  segments3d(matlines)
}
}

require("rgl")
plot3d(reg2)

rgl.snapshot("lm2vue1.png")
par3d(userMatrix=rotate3d(par3d("userMatrix"),-pi*.1, 0, 0, 1))
rgl.close()

```

Solutions to Worksheet from Chap. 9

Managing and Creating Packages

A- Using the Functions **attach()** and **detach()**

- 9.1- Download the file "<http://www.biostatisticien.eu/springeR/bmichild.xls>", using your favourite browser, in your working folder.
- 9.2- Install the package gdata. Since this package necessitates PERL, install also PERL (using "<http://www.biostatisticien.eu/springeR/Rtools.exe>").

```

require("gdata")
bmichild <- read.xls("bmichild.xls")
names(bmichild)

```

- 9.3- When you type GENDER, R answers Error: object 'GENDER' not found.
- 9.4- When you type ls(), you cannot see the variable GENDER.
- 9.5- attach(bmichild)
GENDER

The contents of object GENDER appear.

9.6- The variable GENDER is still not visible with the instruction `ls()`.

9.7- search()

We notice that the object `bmichild` appears in position 2.

9.8- ls(pos=2)

9.9- detach(bmichild)
search()
GENDER

9.10- GENDER <- "Male"
GENDER

9.11- attach(bmichild)

A warning message displays.

GENDER

It is "Male" that displays but not the contents of the object GENDER from the *data.frame*.

9.12- GENDER # does not display the contents of the object
GENDER from the data.frame.
weight

9.13- ls() # bmichild and GENDER are present.
search() # bmichild is present.

9.14- ls(pos=2)

9.15- get("GENDER",pos=2)
bmichild[, "GENDER"]
or
rm(GENDER)
GENDER
Note the existence of the following function:
browseEnv()

B- Creating a Mini-package

- Objects in the Package

9.1- setwd(choose.dir())
alternatively, under Linux:
library("tcltk")
setwd(tk_choose.dir())

9.2- Create the two functions and the data sets:

```
f <- function(x,y) x+y
g <- function(x,y) x-y
d <- data.frame(a=1,b=2)
e <- rnorm(1000)
```

9.3- package.skeleton(name="SmallRPkg",list=c("f","g","d","e"))

9.4- Effective creation of the file of the package. You must modify the help files .Rd.

```
file.show(file.path(R.home("doc"), "KEYWORDS"))
```

9.5- Modify the file DESCRIPTION.

9.6- Read then delete the file Read-and-delete-me.

9.7- Checking (and possibly modification) of the PATH environment variable.

9.8- In a DOS command window, type the following instructions:

```
# Change to the folder containing your package:
cd C:\Documents and Settings\johndoe\Desktop
R CMD check SmallRPkg
R CMD INSTALL --build SmallRPkg
```

9.9- Install your package SmallRPkg.zip. For this, type, for example,

```
help(package="SmallRPkg")
help(d)
```

Solutions to Worksheet from Chap. 10

Matrix Operations, Optimization and Integration

A- The First Optimization Problem

10.1- A <- matrix(c(2,3,5,4),nrow=2,ncol=2)
myf <- function(x) {
 res <- det(A-x*diag(2))
 return(res)
}

10.2- myf <- function(x) {
 n <- length(x)
 res <- rep(NA,n)
 for (i in 1:n) res[i] <- det(A-x[i])*diag(2))

```

    return(res)
}

```

10.3- `curve(myf,xlim=c(-10,10))
abline(h=0,v=0)`

Note that we can use the instruction `locator(2)$x` to “find” the two roots.

10.4- `uniroot(myf,lower=-5,upper=0)$root
uniroot(myf,lower=0,upper=10)$root`

10.5- The polynomial can be written $P(x) = -7 - 6x + x^2$. Its roots are obtained using

```
polyroot(c(-7,-6,1))
```

10.6- `eigen(A)$values`

B- The Second Optimization Problem

10.1- To simplify things, we do not reproduce the figure at the correct scale.

```

plot.new()
par(xpd=NA)
rect(0,0,1,1)
points(0,1,pch=16)
text(0,1,"A",adj=c(2,-0.1),col="blue")
points(1,1,pch=16)
text(1,1,"B",adj=c(-1,0),col="blue")
points(1,0,pch=16)
text(1,0,"C",adj=c(-0.5,1),col="blue")
points(0,0,pch=16)
text(0,0,"D",adj=c(1.7,1),col="blue")
points(0.5,0,pch=16)
text(0.5,0,"H",adj=c(1.5,-0.5),col="blue")
points(1,0.4,pch=16)
text(1,0.4,"Q",pos=4,col="blue")
points(0.5,0.4,pch=16)
text(0.5,0.4,"M",adj=c(0.5,-1),col="blue")
segments(0.5,0,0.5,0.4)
segments(0.5,0.4,1,0.4)
polygon(x=c(0.5,0,1),y=c(0.4,1,1),
        col=rgb(t(col2rgb("brown"))/256,
               alpha=20/100),border="brown")
x <- seq(from=0.6,to=0.565,length=100)
points(x,sqrt(0.1^2-(x-0.5)^2)+0.4,type="l")
text(0.61,0.45,expression(alpha))

```

10.2- We have $\cos(\alpha) = MQ/MB$; hence, $MB = 5/\cos(\alpha)$ and $MA = MB$. $MH = BC - BQ$, $\tan(\alpha) = BQ/MQ = BQ/(AB/2)$ so $BQ = 5\tan(\alpha)$ and $MH = 6 - 5\tan(\alpha)$. Thus $g(\alpha) = 10/\cos(\alpha) + 6 - 5\tan(\alpha) = (10 - 5\sin(\alpha))/\cos(\alpha) + 6 = 5(2 - \sin(\alpha))/\cos(\alpha) + 6$.

10.3- `g <- function(alpha){5*(2-sin(alpha))/cos(alpha)+6}`

10.4- `optimize(g,lower=0,upper=pi/2,tol=0.000001)`

10.5- $g'(\alpha) = 5(-\cos^2(\alpha) - (2 - \sin(\alpha))(-\sin(\alpha)))/\cos^2(\alpha) = 5(-\cos^2(\alpha) + 2\sin(\alpha) - \sin^2(\alpha))/\cos^2(\alpha) = 5(2\sin(\alpha) - 1)/\cos^2(\alpha)$ since $\cos^2(\alpha) + \sin^2(\alpha) = 1$.

10.6- `D(expression(5*(2-sin(alpha))/cos(alpha)+6),"alpha")`

10.7- `gprime <- function(alpha){5*(2*sin(alpha)-1)/ (cos(alpha))^2}`

10.8- `uniroot(gprime,lower=0,upper=pi/2,tol=0.000001)$root`

C- Standard Normal Table

10.1- `phi <- function(x) {exp(-x^2/2)/sqrt(2*pi)}`

10.2- `quantiles <- seq(0,5.5,by=0.01)`
`n <- length(quantiles)`
`probs <- vector(mode="numeric",length=n)`
`for (i in 1:n) probs[i] <- integrate(phi,lower=-Inf,`
`upper=quantiles[i],rel.tol=0.00001)$value`

10.3- `all.equal(probs,pnorm(quantiles))`

10.4- `plot(c(rev(-quantiles),quantiles),c(rev(1-probs),probs),`
`type="l")`

10.5- `curve(pnorm(x),add=TRUE,col="blue")`

D- Principal Components Analysis

10.1- `url <- "http://www.biostatisticien.eu/springeR/`
`climatewine.csv"`
`climatewine <- read.table(url,sep="\t",header=TRUE)`
`attach(climatewine)`
`names(climatewine)`

10.2- `X <- as.matrix(climatewine[,-c(1,6)])`

10.3- `g <- colMeans(X)`
`round(g,2)`

```

10.4- Xdot <- scale(X,scale=FALSE)

10.5- n <- nrow(X)
        inertia <- sum(Xdot^2)/n
        inertia

10.6- inertiacontr <- rowSums(Xdot^2)/n/inertia
        inertiacontr

10.7- onen <- as.matrix(rep(1,n))

10.8- g
        (g <- t(X)%*%onen/n)

10.9- Xdot
        (Xdot <- X-onen%*%t(g))

10.10- (S <- t(Xdot)%*%Xdot/n)
        cov(X)*(n-1)/n

10.11- Doneovers <- diag(1/sqrt(diag(S)))

10.12- Z <- Xdot%*%Doneovers

10.13- t(Z)%*%Z/n
        (R <- cor(X))

10.14- Lambda <- diag(eigen(R)$values)
        W <- eigen(R,symmetric=TRUE)$vectors

10.15- theta <- seq(0,2*pi,.05)
        x <- cos(theta)
        y <- sin(theta)
        plot(x,y,type="l")
        abline(h=0,v=0)
        A <- W%*%Lambda^(1/2)
        text(A[,1:2],colnames(X))
        arrows(rep(0,4),rep(0,4),A[,1],A[,2],length=0.1)

10.16- p <- ncol(X)
        cumsum(diag(Lambda))/p*100

        The first two axes explain 87.6 % of the total inertia.

10.17- CW <- Z%*%W

10.18- dev.new()
        plot(CW[,c(1,2)],type="p",xlab="Axis 1",ylab="Axis 2")
        text(CW[,c(1,2)],labels=YEAR)
        abline(h=0,v=0)

```

```

10.19- plot(CW[,c(1,2)],type="n",xlab="Axis 1",ylab="Axis 2")
good <- CW[QUALITY==1,c(1,2)]
average <- CW[QUALITY==2,c(1,2)]
mediocre <- CW[QUALITY==3,c(1,2)]
text(good,labels=YEAR[QUALITY==1],col="red")
text(average,labels=YEAR[QUALITY==2],col="blue")
text(mediocre,labels=YEAR[QUALITY==3],col="green")
abline(h=0,v=0)
legend("topleft",c("good","average","mediocre"),
      fill=c("red","blue","green"))

10.20- QLT <- apply(sweep(CW^2,1,apply(CW^2,FUN=sum,1),FUN="/")
                      [,1:2],FUN=sum,1)
round(QLT,2)

10.21- require("ade4")

10.22- rownames(X) <- climatewine[,1]
res <- dudi.pca(X) # Enter 2 for the number of axes.
scatter(res)
s.class(res$li,as.factor(climatewine[,6]))

```

Solutions to Worksheet from Chap. 11

Descriptive Data Studies

A- Thoughts on Independence in Descriptive Statistics

```

11.1- url <- "http://www.biostatisticien.eu/springeR/snee74en
          .txt"
snee <-read.table(url,header=TRUE)

```

```

11.2- head(snee)
tail(snee)
dim(snee)
str(snee)

```

There are 592 individuals and three qualitative variables.

```

11.3- attach(snee)
names(snee)
class(hair)
levels(hair)

```

```
class(eyes)
levels(eyes)
class(gender)
levels(gender)
```

11.4- Study of the variable hair:

```
table(hair) # Frequencies.
round(table(hair)/length(hair)*100,2) # Frequencies in
# percentage.
names(which.max(table(hair))) # Mode of the variable.
barplot(sort(table(hair),TRUE),col=c("yellow2","tan4",
"black","tan1"))
```

Study of the variable eyes:

```
table(eyes) # Frequencies.
round(table(eyes)/length(eyes)*100,2) # Frequencies
# in percentage.
names(which.max(table(eyes))) # Mode of the variable.
barplot(sort(table(eyes),TRUE),col=c("blue","brown","tan3",
"green"))
```

Study of the variable gender:

```
table(gender) # Frequencies.
round(table(gender)/length(gender)*100,2) # Frequencies
# in percentage.
names(which.max(table(gender))) # Mode of the variable.
barplot(sort(table(gender),TRUE),col=c("pink","blue"),
pareto=TRUE)
```

11.5- eyeshair <- table(eyes,hair) # Contingency table.

11.6- fhair <- margin.table(eyeshair,2)/592 # Column profiles.
fhair

11.7- # Number of individuals having blue eyes:
nblue <- margin.table(eyeshair,1)[1]
nblue

11.8- round(fhair*nblue)

11.9- marginX <- margin.table(eyeshair,1)
tab.ind1 <- marginX%*%t(fhair)
round(tab.ind1)

11.10- feyes <- margin.table(eyeshair,1)/592 # Row profiles.
feyes

11.11- # Number of individuals having blond hair:
 nblond <- margin.table(eyeshair,2)[1]
 nblond

11.12- round(feyes*nblond)

11.13- marginY <- margin.table(eyeshair,2)
 tab.ind2 <- feyes%*%t(marginY)
 round(tab.ind2)

11.14- all.equal(tab.ind1,tab.ind2)

The two tables are identical, which is reassuring since interest in the independence of the eyes and hair is equivalent to an interest in the independence of the hair and eyes.

11.15- (eyeshair-tab.ind1)^2

11.16- tab.contr <- (eyeshair-tab.ind1)^2/tab.ind1
 tab.contr

11.17- chi2 <- sum(tab.contr)
 chi2

Phi2 <- chi2/sum(eyeshair)
 Phi2

C <- sqrt(chi2/(sum(eyeshair)+chi2))
 C

V2 <- Phi2/(min(dim(eyeshair))-1)
 V2

All these indicators are not zero, so there is some form of dependence in this population (studied under the descriptive statistics framework).

11.18- # Conditional distributions of the variable hair
 # given that eyes = (blue, or brown or ...):
 prop.table(eyeshair,1)

Rows are not identical. This confirms the dependency (from a descriptive statistics perspective) between the two variables.

Conditional distributions of the variable eyes
 # given that hair = (blond or ...):
 prop.table(eyeshair,2)

Columns are not identical. This also confirms the dependency (from a descriptive statistics perspective) between the two variables.

11.19- `table(hair,gender)`
`plot(hair~gender)`
`plot(table(hair,gender))`

11.20- `table(eyes,gender)`
`plot(eyes~gender)`
`plot(table(eyes,gender))`

11.21- See worksheet in Chap. 2 to know how to import the table, then follow the steps above.

B- Descriptive Analysis of Data Set NUTRIELDERLY

11.1- `require("gdata")`
`url <- "http://www.biostatisticien.eu/springeR/`
`nutrition_elderly.xls"`
`nutrition <- read.xls(url,header=TRUE)`
`attach(nutrition)`

11.2- `names(which.max(table(situation)))`
`names(which.max(table(chocol)))`
`names(which.max(table(height)))`

11.3- `res <- hist(height,breaks=seq(140,190,by=5),right=TRUE,`
`plot=FALSE)`
`ind <- which.max(res$count)`
`modal.class <- paste(res$breaks[ind],res$breaks[ind+1],`
`sep="-")`

The modal class is the class]155; 160].

11.4- `median(chocol)`

Note that if the studied variable is considered as an ordinal variable, then the function `median()` of R does not work anymore:

`median(as.ordered(chocol))`

But we can propose a self-made function to realize this operation:

```
my.median <- function(x) {
  if (is.numeric(x)) return(median(x))
  if (is.ordered(x)) {
    N <- length(x)
    if (N%%2) return(sort(x)[(N+1)/2]) else {
      inf <- sort(x)[N/2]
      sup <- sort(x)[N/2+1]
      if (inf==sup) return(inf) else return(list(inf,sup))}}
```

```

        }
      stop("Computation of the median not possible for
            this type.")
    }
  my.median(as.ordered(chocol))
}

```

11.5- `table(chocol)`
`table(raw_fruit)`

11.6- We can use the following self-made function:

```

med <- function(tabx) names(which.max(cumsum(tabx)/
                     sum(tabx)>0.5))
med(table(chocol))
med(table(raw_fruit))

```

11.7- `quartile.on.freq <- function(tabx, quart) {`
 `# tabx is the table of frequencies.`
 `tab.freq.cum <- cumsum(tabx)/sum(tabx)`
 `index <- order(tab.freq.cum < quart)[1]`
 `f1 <- tab.freq.cum[index]`
 `f2 <- tab.freq.cum[index-1]`
 `x1 <- as.numeric(names(f1))`
 `x2 <- as.numeric(names(f2))`
 `quartile <- as.numeric(x1 + (x2-x1)*(quart-f1)/(f2-f1))`
 `return(quartile)`
`}`

```

tab <- res$counts
names(tab) <- res$breaks[-1]

quartile.on.freq(tab,0.25)
quartile.on.freq(tab,0.5)
quartile.on.freq(tab,0.75)

```

11.8- `breaks <- res$breaks`
`plot(breaks,ecdf(height)(breaks),type="l",main=`
 `paste("Cumulated frequency polygon",`
 `"of variable height",sep="\n"),ylab="Frequencies",`
 `col="darkolivegreen",lwd=3)`
`abline(h=c(0.25,0.5,0.75))`
`locator(1)$x # Click on the searched intersection.`

11.9- `mean(height)`
`mean(weight)`
`mean(age)`

```

11.10- table(tea)
sum(c(0:6,9,10)*as.numeric(table(tea)))/sum(table(tea))

11.11- sum(res$mids*res$counts)/sum(res$counts)

11.12- diff(range(weight))

11.13- boxplot(weight)

11.14- var.pop <- function(x) var(x)*(length(x)-1)/length(x)
sd.pop <- function(x) sqrt(var.pop(x))
sd.pop(height)

11.15- coeffvar.tab <- function(tabx) {
  val <- as.numeric(names(tabx))
  freq <- as.numeric(tabx)/sum(tabx)
  mean <- sum(val*freq)
  var <- sum(val^2*freq) - mean^2
  res <- sqrt(var)/mean
  return(res)
}
coeffvar.tab(table(tea))

11.16- eta2 <- function(x,gp) {
  means <- tapply(x,gp,mean)
  frequencies <- tapply(x,gp, length)
  varinter <- (sum(frequencies * (means - mean(x))^2))
  vartot <- (var(x) * (length(x) - 1))
  res <- varinter/vartot
  list(var.tot=vartot,var.inter=varinter,
       var.intra=vartot-varinter,eta2=res)
}

res <- eta2(tea,gender)

```

Solutions to Worksheet from Chap. 12

Simulations

A- Study of the Distribution $f(x) = \frac{3}{2} \sqrt{x}$ on $[0, 1]$

```

12.1- f <- function(x) (3/2)*sqrt(x)
integrate(f,lower=0,upper=1)

```

12.2- Method of generic inversion:

```
Finv <- function(x) x^(2/3)
u <- runif(1000)
x <- Finv(u)
```

12.3- mean(x)
var(x)

12.4- Theoretical values are $E(X) = 3/5$ and $\text{Var}(X) = 12/175$.

12.5- The cumulative distribution function is $F(x) = x^{2/3}$. The theoretical probabilities of the various classes are: $0.3^{3/2} = 0.1643168$, $0.5^{3/2} - 0.3^{3/2} = 0.1892366$, $0.7^{3/2} - 0.5^{3/2} = 0.2321086$, $0.85^{3/2} - 0.7^{3/2} = 0.1979993$ and $1 - 0.85^{3/2} = 0.2163387$. We can evaluate them numerically:

```
class <- c(0.3,0.5,0.7,0.85)
prob <- function(x) {integrate(f,lower=0,upper=x)$value}
dist <- c(0,apply(as.matrix(class),MARGIN=1,FUN=prob),1)
(theoretical.prob <- diff(dist))
```

The empirical probabilities can be computed using the following code:

```
count <- function(x,vect.y) sum(vect.y<x)/length(vect.y)
empirical.dist <- c(0,apply(as.matrix(class),MARGIN=1,
                           FUN=count,x),1)
(empirical.prob <- diff(empirical.dist))
```

or the following code:

```
c(mean(0<=x & x<=0.3),mean(0.3<x & x<=0.5),mean(0.5<x &
  x<=0.7),
  mean(0.7<x & x<=0.85),mean(0.85<x & x<=1))
```

B- Study of the Generalized Pareto Distribution

12.1- rGP <- function(n,mu,sigma,xi)mu
+sigma*((runif(n))^(-xi)-1)/xi

12.2- n <- 1000
simu <- rGP(n,0,1,0.25)

12.3- mean(simu)
var(simu)

12.4- mu <- 0
sigma <- 1
xi <- 0.25
(theo.mu <- mu + sigma/(1-xi))
(theo.var <- (sigma^2)/(((1-xi)^2)*(1-2*xi)))

```

12.5- n <- 10000
  simu.new <- rGP(n, 0, 1, 0.25)
  mean(simu.new)
  var(simu.new)

12.6- hist(simu.new, breaks=500, xlim=c(0, 10), prob=TRUE, col="red")

12.7- dens.pareto <- function(x, mu, sigma, xi) {
  (1/sigma)*((1+(xi*(x-mu))/sigma)^(-1/xi-1))
}
curve(dens.pareto(x, mu, sigma, xi), add=TRUE, col="blue")

```

C- Uniform Distribution on a Square

```

12.1- n <- 1000
  simu <- data.frame(X1=runif(n), X2=runif(n))

12.2- The distance of  $(X_1, X_2)$  to the nearest edge is given by  $D = \min(X_1, X_2, 1 - X_1, 1 - X_2)$  (make a drawing to be convinced).
  simu.d <- cbind(simu, 1-simu)
  D <- apply(simu.d, MARGIN=1, FUN=min)
  mean(D<0.25)

12.3- dist.sum <- function(coord){
  d <- min(coord[1]^2+coord[2]^2, coord[1]^2+coord[4]^2,
            coord[3]^2+coord[2]^2, coord[3]^2+coord[4]^2)
  d <- sqrt(d)
}

D.sum <- apply(simu.d, MARGIN=1, FUN=dist.sum)
mean(D.sum<0.25)

12.4- mean(D)
  var(D)
  hist(D, prob=TRUE)
  curve(-8*x+4, add=TRUE)
  # Density on [0, 0.5] : f(x)=-8x+4

```

D- Towards Modelling

```

12.1- X <- function() ifelse(rbinom(1, 1, 0.5), "HEADS", "TAILS")
  X()

12.2- throw.of.a.die <- function() sample(1:6, 1)
  throw.of.a.die()

```

12.3- `yams <- function() sample(1:6,size=5,replace=TRUE)`

12.4- `n <- 1000000`

```
result <- replicate(n,yams(),TRUE)
test <- function(res) ifelse(length(unique(res))> 1,0,1)
prop <- mean(apply(result,MARGIN=2,FUN=test))
```

E- Box–Muller Theorem

12.1- `n <- 1000`

```
u1 <- runif(n)
u2 <- runif(n)
z1 <- sqrt(-2*log(u1))*cos(2*pi*u2)
z2 <- sqrt(-2*log(u1))*sin(2*pi*u2)
```

12.2- Non-parametric estimation of the density:

```
require("MASS")
ngrid <- 40
denobj<-kde2d(z1,z2,n=ngrid)
den.z <-denobj$z
```

12.3- Surface of the density of the bivariate normal:

```
require("rgl")
xgrid <- denobj$x
ygrid <- denobj$y
bi.z <- dnorm(xgrid)%%t(dnorm(ygrid))
```

New window:

```
open3d()
bg3d(color="#887777")
light3d()
```

Representing simulated data:

```
spheres3d(z1,z2,rep(0,n),radius=0.1,color="#CCCCFF")
```

Representing the non-parametric estimated density:

```
surface3d(xgrid,ygrid,den.z*20,color="#FF2222",alpha=0.5)
```

Representing the standard bivariate normal:

```
surface3d(xgrid,ygrid,bi.z*20,color="#CCCCFF",
          front="lines")
```

Solutions to Worksheet from Chap. 13

A- Study of Confidence Intervals

- Study of the confidence interval for the mean

13.1- `n <- 20`

```

M <- 50000
ech <- rnorm(n,mean=-1.2,sd=sqrt(2)) # Only a sample.
simu <- replicate(M,rnorm(n,mean=-1.2,sd=sqrt(2)),
simplify=TRUE)

```

13.2- `inter.mu <- function(x) t.test(x,conf.level=0.9)$conf.int`
`res <- t(apply(simu,MARGIN=2,FUN=inter.mu))`

13.3- `prop <- mean((res[,1] < -1.2) & (-1.2 < res[,2]))`

Around 90 % of the intervals contain the value -1.2 .

13.4- `n <- 100`

```

M <- 50000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
res.chi2 <- t(apply(simu,MARGIN=2,FUN=inter.mu))
prop <- mean((res.chi2[,1] < 1) & (1 < res.chi2[,2]))

```

13.5- `n <- 10`

```

M <- 50000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
res.chi2 <- t(apply(simu,MARGIN=2,FUN=inter.mu))
prop <- mean((res.chi2[,1] < 1) & (1 < res.chi2[,2]))

```

The (empirical) level of confidence of the interval is not equal to 90 %. This is due to the small sample size ($n = 10$), which is not sufficient to use the central limit theorem.

13.6- `n <- 20`

```

ech.20 <- rnorm(n,mean=-1.2,sd=sqrt(2))
inter.mu <- function(x) t.test(x,conf.level=0.95)$conf.int
inter.mu(ech.20)

```

13.7- `ech.50 <- rnorm(50,mean=-1.2,sd=sqrt(2))`

```

inter.mu(ech.50)
ech.100 <- rnorm(100,mean=-1.2,sd=sqrt(2))
inter.mu(ech.100)
ech.1000 <- rnorm(1000,mean=-1.2,sd=sqrt(2))
inter.mu(ech.1000)
ech.10000 <- rnorm(10000,mean=-1.2,sd=sqrt(2))

```

```
inter.mu(ech.10000)
ech.100000 <- rnorm(100000,mean=-1.2,sd=sqrt(2))
inter.mu(ech.100000)
```

The confidence interval gets narrower around the true value of μ .

13.8-

```
tt.20 <- c(t.test(ech.20)$st,t.test(ech.20)$p.val)
tt.50 <- c(t.test(ech.50)$st,t.test(ech.50)$p.val)
tt.100 <- c(t.test(ech.100)$st,t.test(ech.100)$p.val)
tt.1000 <- c(t.test(ech.1000)$st,t.test(ech.1000)$p.val)
tt.10000 <- c(t.test(ech.10000)$st,t.test(ech.10000)
                $p.val)
tt.100000 <- c(t.test(ech.100000)$st,t.test(ech.100000)
                $p.val)
output <- rbind(tt.20,tt.50,tt.100,tt.1000,tt.10000,
                  tt.100000)
colnames(output) <- c("tobs","pvalue")
output
```

As n increases, t_{obs} increases and the p -value gets smaller. It becomes more and more difficult to “show” that μ is different from 0.

13.9-

```
inter.mu(ech.100000)
t.test(ech.100000, mu=-1.1)$p.val
tt.50
```

The p -value is smaller here although -1.1 is nearer from the true μ ($= -1.2$) than the reference value 0 used at the previous question. On the other side, the confidence interval does not deceive. We see the interest of confidence intervals as compared to only giving the p -value!

- Study of confidence intervals from bootstrap

13.1-

```
n <- 20
M <- 500
ech <- rexp(n,rate=1/10) # only one sample
simu.exp <- replicate(M,rexp(n,rate=1/10),simplify=TRUE)
```

13.2- For one sample:

```
require("boot")
mean <- function(x,indices) mean(x[indices])
mean.boot <- boot(simu.exp[,3],mean,R=999,stype="i",
                  sim="ordinary")
boot.ci(mean.boot,conf=0.9,type="perc")$percent[4:5]
```

For M samples:

```
inter.boot.mean <- function(y) {
  mean <- boot(y,mean,R=999,stype="i",sim="ordinary")
  boot.ci(mean,conf=0.9,type="perc")$percent[4:5]
}
```

```
res <- t(apply(simu.exp,MARGIN=2,FUN=inter.boot.mean))
```

13.3- prop.boot <- mean((res[,1] < 10) & (res [,2] > 10))

13.4- res.ttest <- t(apply(simu.exp,MARGIN=2,FUN=inter.mu))
prop.ttest <- mean((res.ttest[,1] < 10) & (res.ttest [,2] > 10))

B- Study of Risks in Hypothesis Testing

- Study of risk of the first kind

13.1- n <- 20
M <- 500
ech <- rnorm(n,mean=4,sd=sqrt(1.2)) # Only one sample.
simu <- replicate(M,rnorm(n,mean=4,sd=sqrt(1.2)),
simplify=TRUE)

13.2- ttest <- function(x) t.test(x, mu=4)\$p.value
res <- apply(simu,MARGIN=2,FUN=ttest)

13.3- (count <- sum(res < 0.05))

count hypotheses are rejected; we expect to reject 25 null hypotheses.

13.4- M <- 5000
simu <- replicate(M,rnorm(n,mean=4,sd=sqrt(1.2)),
simplify=TRUE)
res <- apply(simu,MARGIN=2,FUN=ttest)
proportion <- mean(res < 0.05)
proportion

13.5- n <- 100
M <- 5000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
ttest <- function(x) t.test(x, mu=1)\$p.value
res.chi2 <- t(apply(simu,MARGIN=2,FUN=ttest))
proportion <- mean(res.chi2 < 0.05)
proportion

```
13.6- n <- 10
M <- 5000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
ttest <- function(x) t.test(x,mu=1)$p.value
res.chi2 <- t(apply(simu,MARGIN=2,FUN=ttest))
proportion <- mean(res.chi2 < 0.05)
proportion
```

The risk of the first kind of the test is greater than 5 % because the Student test is not valid here due to a small sample size.

```
13.7- IC.p <- t.test((res.chi2<0.05)*1)$conf
IC.p
# or also:
binom.test(sum(res.chi2<0.05),length(res.chi2))$conf
```

- Study of power

```
13.1- n <- 20
M <- 500
simu <- replicate(M,rnorm(n,mean=5,sd=sqrt(1.2)),
simplify=TRUE)
```

```
13.2- ttest <- function(x) t.test(x,mu=4)$p.value
res <- apply(simu,MARGIN=2,FUN=ttest)
```

```
13.3- count <- sum(res < 0.05)
power <- mean(res < 0.05)
```

```
13.4- n <- 100
M <- 500
simu <- replicate(M,rnorm(n,mean=5,sd=sqrt(1.2)),
simplify=TRUE)
res <- apply(simu,MARGIN=2,FUN=ttest)
(power <- mean(res < 0.05))
```

The power is 100 %. It has increased with the sample size.

```
13.5- n <- 100
M <- 500
simu <- replicate(M,rchisq(n,df=2),simplify=TRUE)
ttest <- function(x) t.test(x,mu=1)$p.value
res.chi2 <- apply(simu,MARGIN=2,FUN=ttest)
proportion <- mean(res.chi2 < 0.05)
proportion
```

```
13.6- n <- 10
M <- 500
simu <- replicate(M,rchisq(n,df=2),simplify=TRUE)
```

```
ttest <- function(x) t.test(x, mu=1)$p.value
res.chi2 <- t(apply(simu, MARGIN=2, FUN=ttest))
proportion <- mean(res.chi2 < 0.05)
proportion
```

The power of the test is low due to a small sample size $n = 10$. We do not have enough information to show that the mean is different from 1.

C- A Few Practical Examples

- Cow study

```
13.1- just.after.milking <- c(12000,13000,21500,17000,15000,
                               22000,11000,21000)
after.milking.24h <- c(11000,20000,31000,28000,26000,
                      30000,16000,29000)
t.test(just.after.milking, after.milking.24h, paired=TRUE,
       alt="less")

13.2- # Test of signs:
dif <- just.after.milking - after.milking.24h
sminus <- sum(dif<0)
splus <- sum(dif>0)
binom.test(min(splus,sminus), splus+sminus, alt="less")

13.3- # Mann-Whitney's test:
wilcox.test(just.after.milking, after.milking.24h,
            paired=TRUE, exact=FALSE, alt="less")
```

- East German athletes

```
13.1- hormons <- c(3.22,3.07,3.17,2.91,3.4,3.58,3.23,3.11,3.62)
t.test(hormons, mu=3.1, alt="greater")

13.2- The average amount of androgens for athletes in East Germany is significantly higher than the average reference 3.1. They are doped.
```

- Drinking and driving

```
before <- c(57,54,62,64,71,65,70,75,68,70,77,74,80,83)
after <- c(55,60,68,69,70,73,74,74,75,76,76,78,81,90)
t.test(before, after, paired=TRUE)
```

- Speed of light

```
13.1- speed <- c(850,740,900,1070,930,850,950,980,980,880,1000,
               980,930,1050,960,810,1000,1000,960,960)
speed <- speed + 299000
hist(speed)
```

There is a very visible absolute mode.

13.2- shapiro.test(speed)

13.3- t.test(speed,mu=299990)

13.4- mean(speed)

- Cholesterol levels

```
groupA <- c(46.3,42.5,43,43.9,42,41.5,41.6,44.4,40.7)
groupB <- c(47.1,44.5,45.8,49,44.6,43.7,44.5,47.4)
t.test(groupA,groupB,alt="less")
```

We can conclude that the treatment is efficient at significance level $\alpha = 5\%$.

- Treatment–death independence

```
x <- matrix(c(0,9,8,3),ncol=2,byrow=TRUE)
# Table of theoretical frequencies:
chisq.test(x)$exp # Some values are < 5.

chisq.test(x) # Yates chi-2.
fisher.test(x) # Fisher's exact test.
```

Mortality depends on the treatment.

- Number of patients in the emergency ward

```
files <- c(1500,1600,1450)
cases <- c(675,720,610)
chisq.test(cases/files)
```

In view of the results, we cannot conclude that the proportion of emergencies is different depending on the month.

Solutions to Worksheet from Chap. 14

A- Study of Simple Linear Regression

- Study of synthetic data

14.1- n <- 50

b0 <- 1

b1 <- 2

sigma <- 0.1

t <- 5

error <- rnorm(n, sd=sigma)

x <- runif(n, min=0, max=t)

y <- b0+b1*x+error

14.2- plot(y~x)

14.3- model <- lm(y~x)

coefficients(model)

sd(residuals(model))*sqrt((n-1)/(n-2))

or also:

summary(model)[6]

14.4- plot(model, 1:2)

plot(model\$fitted~y)

14.5- precision.parameter.b1 <- function(n, sigma) {

 error <- rnorm(n, sd=sigma)

 x <- runif(n, min=0, max=t)

 y <- b0 + b1*x + error

 model <- lm(y~x)

 summary(model)\$coef[2,2]

}

size <- seq(50, 1000, by=50)

sd.b1 <- vector("numeric", 20)

for (i in 1:20) {

 sd.b1[i] <- precision.parameter.b1(size[i], 0.1)

}

We can see on the graph below the precision of the estimate of b1 according to n for a noise standard deviation equal to 0.1:

```
plot(sd.b1~size,xlab="n",
      ylab="Estimation of standard deviation of the
            estimator of b1",
      main="Precision of the estimation of b1 as
            a function of n")
```

On the chart below, we can see the accuracy of the estimate of b_1 according to σ for n equal to 100:

```
sd <- seq(0.1,10,by=0.1)
sd.b1 <- vector("numeric",100)
for (i in 1:100) {
  sd.b1[i] <- precision.parameter.b1(n=100, sd[i])
}

title <- "Precision of the estimation of b1 as a function
          \n of the standard deviation of the noise"
plot(sd.b1~sd, xlab=expression(sigma), ylab=
"estimation of the standard deviation of the estimator
  of b1", main=title)
```

- Study of intima–media

14.1- `url <- "http://biostatisticien.eu/springeR/Intima_Media_Thickness.xls"`
`require("gdata")`
`intima.media <- read.xls(url,header=TRUE)`
`attach(intima.media)`

14.2- `plot(measure~AGE)`

Increasing trend of the measure of the thickness of the intima–media as a function of AGE.

14.3- `# Calculating the correlation coefficient between the two variables:`
`cor(measure,AGE)`
`# Test the significance of the coefficient of correlation:`
`cor.test(measure,AGE)`

14.4- `model <- lm(measure~AGE,data=data.frame(measure,AGE))`
`summary(model)`
`abline(model,col="blue")`

14.5- `hist(residuals(model),main="Histogram")`
`plot(model,1:6)`

```

14.6- age0 <- 33
predict(model,data.frame(AGE=age0),interval="prediction")

14.7- predict(model,data.frame(AGE=age0),interval="confidence")

14.8- model2 <- lm(measure~I(AGE^2))
summary(model2)

```

The R^2 increased (a little bit).

B- Study of Multiple Linear Regression

- Study of intima–media

We notice (tobacco[is.na(packyear)]) and packyear[tobacco==0]) that the variable packyear (giving the number of pack of cigarettes per year) contains missing values NA each time the variable tobacco takes the value 0 (meaning a non smoker). It is thus natural to replace missing data of variable packyear with zeroes:

```
intima.media$packyear[is.na(packyear)] <- 0
```

Creation of the variable BMI:

```
BMI <- intima.media$weight/((intima.media$height/100)^2)
attach(intima.media)
my.data <- data.frame(AGE,SPORT,alcohol,packyear,BMI,measure)
```

```
14.1- pairs(my.data)
```

Some scatter plots have a linear trend, which can lead to collinearity.

```

14.2- model.age <- lm(measure~AGE,data=my.data)
summary(model.age)

model.sport <- lm(measure~SPORT,data=my.data)
summary(model.sport) ### p-value > 0.25

model.alcohol <- lm(measure~factor(alcohol),data=my.data)
summary(model.alcohol)

model.packyear <- lm(measure~packyear,data=my.data)
summary(model.packyear)

model.BMI <- lm(measure~BMI,data=my.data)
summary(model.BMI)

14.3- model.age.inter <- lm(measure~AGE*packyear)
# Non-significant interaction at 0.25 level:
summary(model.age.inter)

model.alcohol.without.inter <-
lm(measure~factor(alcohol)+packyear,data=my.data)

```

```

model.alcohol.inter <-
  lm(measure~factor(alcohol)*packyear,data=my.data)
summary(model.alcohol.inter)
anova(model.alcohol.without.inter,model.alcohol.inter)

model.BMI.inter <- lm(measure~BMI*packyear,data=my.data)
# Non-significant interaction at level 0.25:
summary(model.BMI.inter)

14.4- model.inter <-
  lm(measure~AGE+factor(alcohol)*packyear+BMI,
     data=my.data)
summary(model.inter)

14.5- model.without.inter <- lm(measure~AGE+factor(alcohol)+
  packyear+BMI,data=my.data)
# The interaction term is not significant anymore:
anova(model.inter,model.without.inter)
summary(model.without.inter)

14.6- modele.without.alcohol <- lm(measure~AGE+packyear+BMI,
  data=my.data)
anova(model.without.alcohol,model.without.inter)

```

The variable alcohol does not bring any information.

```
summary(model.without.alcohol)
```

- 14.7-** In the final model, the measurement of intima–media is explained by the variables AGE and BMI, with an adjustment on the variable packyear. Whatever the number of packs of cigarettes smoked each year, the measurement of intima–media increases with age and BMI.

- Study of unemployment rates

- 14.1-** Download the file "<http://biostatisticien.eu/springeR/unemployment.RData>" in your working folder:

```

load("unemployment.RData")
names(unemployment)
attach(unemployment)

model.txpib <- lm(unemp~gdprate,data=unemployment)
summary(model.gdprate)
plot(unemp~gdprate)
abline(model.gdprate)

```

- ```

hist(residuals(model.gdprate))
plot(model.gdprate,1:2)

14.2- cor(unemp[,2:7])

14.3- pairs(unemp[,2:7])

14.4- Most explicative variables: taxb, govspend, gdprate. Colinear variables:
govspend and taxb, as well as govspend and gdprate, and finally taxb
and gdprate.

14.5- full.model <- lm(unemp~gdprate+govspend+taxb+salav+infl,
 data=unemployment)
summary(full.model)

14.6- require("car")
vif(full.model)

14.7- drop1(lm(unemp~gdprate+govspend+taxb+salav+infl,
 data=unemployment),test="F")
We remove variable infl.
drop1(lm(unemp~gdprate+govspend+taxb+salaav,
 data=unemployment),test="F")
We remove variable txpib.
drop1(lm(unemp~govspend+taxb+salav,data=unemployment),
 test="F")
We remove variable pfisc.
drop1(lm(unemp~govspend+salav,data=unemployment),test="F")

14.8- # Final model:
final.model <- lm(unemp~govspend+salav,data=unemployment)
summary(final.model)

14.9- govspend93 <- unemployment$govspend[unemployment$year ==
 1993]
salav93 <- unemployment$salav[unemployment$year==1993]
predict(final.model,data.frame(govspend=govspend93,
 salav=salav93),
 interval="prediction")

14.10- The observed value of unemployment in 1993 was 11.2 and this value is
contained in the prediction interval.

```

## C- Study of Polynomial Regression

- Study of synthetic data

```

14.1- n <- 100
x <- runif(n,min=-2,max=2)
eps <- rnorm(n)
y <- x+2*(x^2)+3.5*(x^3)-2.3*(x^4)+eps

14.2- plot(y~x)
curve(x+2*x*x+3.5*x^3-2.3*x^4,add=TRUE)

14.3- simple.model <- lm(y~x)
summary(simple.model)
hist(residuals(simple.model))
plot(simple.model,1:6)

14.4- poly.model <- lm(y~-1+poly(x,4,raw=TRUE))
summary(poly.model)
coef <- coef(poly.model)
plot(y~x)
curve(coef[1]*x+coef[2]*(x^2)+coef[3]*x^3+coef[4]*x^4,
 add=TRUE)

```

- Fitting a scatter plot with a polynomial

```

14.1- Download the file http://biostatisticien.eu/springeR/fitpoly.RData in your current directory.

load("fitpoly.RData")
attach(fitpoly)

14.2- plot(Y~X)

14.3- lin.model <- lm(Y~X)
summary(lin.model)
abline(lin.model)

14.4- poly.model <- lm(Y~poly(X,3,raw=TRUE),data=fitpoly)
summary(poly.model)

14.5- coef <- coef(poly.model)
curve(coef[1]+coef[2]*x+coef[3]*x^2+coef[4]*x^3,add=TRUE)
x <- seq(-3.5,3.5,length=50)
pred.int <- predict(poly.model,data.frame(X=x),
interval="prediction")[,c("lwr","upr")]

```

```

conf.int <- predict(poly.model,data.frame(X=x),
 interval="confidence")[,c("lwr","upr")]
matlines(x,cbind(conf.int,pred.int),lty=c(2,2,3,3),
 col=c("red","red","blue","blue"),lwd=c(2,2,1,1))
legend("bottomright",lty=c(2,3),lwd=c(2,1),
 c("confidence","prediction"),col=c("red","blue"))

```

## Solutions to Worksheet from Chap. 15

### A- Study of One-Way ANOVA

- Study of noise levels

**15.1-** `noise <- gl(4,6,24)`  
`grade <- c(62,60,63,59,63,59,56,62,60,61,63,64,63,67,`  
`71,64,65,66,68,66,71,67,68,68)`

**15.2-**  $\text{grade}_{ik} = \mu + \alpha_i + \epsilon_{ik}$  where the  $\epsilon_{ik}$ 's are *i.i.d.* random variables following a  $\mathcal{N}(0, \sigma^2)$  distribution.

**15.3-** `model <- aov(grade~noise)`  
`summary(model)`

**15.4-** `pairwise.t.test(grade,nois,p.adjust="bonf")`  
`TukeyHSD(model)`

- Study of intima–media

**15.1-** `url <- "http://biostatisticien.eu/springeR/Intima_Media_Thickness.xls"`  
`require("gdata")`  
`intima.media <- read.xls(url,header=TRUE)`  
`attach(intima.media)`

**15.2-** `plot(measure~factor(alcohol))`

**15.3-** `intima.model <- aov(measure~factor(alcohol))`  
`summary(intima.model)`

There exists a difference in the measure of thickness of intima–media depending on alcohol consumption (p-value  $\leq 0.05$ ).

```
15.4- plot(intima.model)
Homoscedasticity :
require("car")
leveneTest(measure,factor(alcohol))
```

- Study of physical activity

**15.1-** Download the file "<http://biostatisticien.eu/springeR/sports.RData>" in your working folder.

```
load("sports.RData")
attach(sports)
```

Factor time of practicing sport (7 levels).  $\text{score}_{ik} = \mu + \alpha_i + \epsilon_{ik}$  where the  $\epsilon_{ik}$ 's are *i.i.d.* random variables following a  $\mathcal{N}(0, \sigma^2)$  distribution.

**15.2-** sport.model <- aov(score~time)

```
summary(sport,model)
```

**15.3-** cmat <- rbind(" : not athletic versus somewhat athletic"=
c(1,1,-1,-1,0,0,0),
" : very athletic versus not athletic"=c(3,3,0,0,-2,-2,-2))

**15.4-** require("gmodels")
fit.contrast(sport.model,time,cmat)

## B- Study of Two-Way ANOVA

- Study of batteries

**15.1-** Factor temperature with three levels: 15, 70 and 125 degrees. Factor type with three levels: I, II and III. Dependent variable: lifetime.

**15.2-**  $Y_{ijk} = \mu + \mu_{ij} + \epsilon_{ijk}$  where the  $\epsilon_{ijk}$ 's are *i.i.d.* random variables following a  $\mathcal{N}(0, \sigma^2)$  distribution.

```
15.3- lifetime <- c(130,155,74,180,34,40,80,75,20,70,82,58,150,
188,159,126,136,122,106,115,25,70,58,45,138,
110,168,160,174,120,150,139,96,104,82,60)
temperature <- as.factor(rep(rep(1:3,each=4,3)))
levels(temperature) <- c("15$\circ C", "70$\circ C",
125$\circ C")
type.battery <- as.factor(rep(1:3,each=12))
levels(type.battery) <- c("type I","type II","type III")
battery <- data.frame(lifetime=lifetime,temperature=
temperature,type.battery=type.battery)
```

```
interaction.plot(temperature,type.battery,lifetime)
interaction.plot(type.battery,temperature,lifetime)
```

**15.4-** summary(lm(lifetime~temperature\*type.battery))

**15.5-** anova(lm(lifetime~temperature\*type.battery))

Significant interaction: be careful about the interpretation of parameters and tests of fixed effects; we must look at the conditional effects.

**15.6-** For example, test of the temperature effect for batteries of type I

```
model <- summary(aov(lifetime~temperature*type.battery))
lifetime.battery1 <- summary(aov(lifetime~temperature,
 subset=type.battery=="type I"))
lifetime.battery1
F.lifetime.battery1 <- lifetime.battery1[[1]]$Mean[1]/
 model[[1]]$Mean[4]
pvalue <- 1-pf(F.lifetime.battery1,df1=2,df2=27)
```

- Milk yield

**15.1-** Factor food with four levels: Straw, Hay, Grass and Silage. Factor dose with two levels: Low and High. Dependent variable: yield.  $Y_{ijk} = \mu + \mu_{ij} + \epsilon_{ijk}$  where the  $\epsilon_{ijk}$ 's are *i.i.d.* random variables following a  $\mathcal{N}(0, \sigma^2)$  distribution.

**15.2-** yield <- c(8,11,11,10,7,12,13,14,11,10,10,12,12,13,14,17,
 13,17,14,13,8,9,8,10,9,10,7,10,12,11,11,9,
 11,11,12,13,12,11,15,14)

```
food <- as.factor(rep(rep(1:4,each=5,2)))
levels(food) <- c("Straw","Hay","Grass","Silage")
dose <- as.factor(rep(1:2,each=20))
levels(dose) <- c("Low", "High")
milk <- data.frame(yield=yield,food=food,dose=dose)
interaction.plot(food,dose,yield)
interaction.plot(dose,food,yield)
```

**15.3-** summary(lm(yield~dose\*food))

**15.4-** anova(lm(yield~dose\*food))

Nonsignificant interaction: we will prefer a simpler model.

**15.5-** anova(lm(yield~dose+food))

- Intima-media study

15.1- url <- "http://biostatisticien.eu/springeR/Intima\_Media\_Thickness.xls"  
intima.media <- read.xls(url,header=TRUE)  
attach(intima.media)

15.2- Two way ANOVA model with an interaction term.

15.3- tobacco <- factor(tobacco)  
alcohol <- factor(alcohol)  
interaction.plot(tobacco,alcohol,measure)  
interaction.plot(alcohol,tobacco,measure)

15.4- require("car")  
intima.model <- Anova(lm(measure~alcohol\*tobacco),  
type="III")  
intima.model  
# non-significant interaction:  
additive.intima.model <- Anova(lm(measure~alcohol+  
tobacco),type="III")  
# alcohol: p-value=0.09 Tobacco: p-value=0,16