An Introduction to R
A Programming Environment for Data Analysis and Graphics
Second Edition, for Version 2.9.0 (released April 2009)

W. N. Venables, D. M. Smith
and the R Development Core Team
Revised and updated for publication by Network Theory Ltd.
# Table of Contents

Publisher’s Foreword ........................................... 1

Authors’ Preface ............................................... 3

1 Introduction and preliminaries ....................... 5
   1.1 The R environment ...................................... 5
   1.2 Related software and documentation ................. 5
   1.3 R and statistics ......................................... 6
   1.4 R and the window system ............................... 6
   1.5 Using R interactively ................................... 6
   1.6 An introductory session ................................ 7
   1.7 Getting help with functions and features .......... 8
   1.8 R commands, case sensitivity, etc. .................. 9
   1.9 Recall and correction of previous commands ....... 10
   1.10 Executing commands from or diverting output to a file .... 11
   1.11 Data permanency and removing objects .......... 11

2 Simple manipulations; numbers and vectors ... 13
   2.1 Vectors and assignment .................................. 13
   2.2 Vector arithmetic ....................................... 14
   2.3 Generating regular sequences ........................ 15
   2.4 Logical vectors ......................................... 16
   2.5 Missing values .......................................... 16
   2.6 Character vectors ....................................... 17
   2.7 Index vectors; selecting and modifying subsets of a data set .... 18
   2.8 Other types of objects .................................. 19

3 Objects, their modes and attributes ............... 21
   3.1 Intrinsic attributes: mode and length ................. 21
   3.2 Changing the length of an object .................... 22
   3.3 Getting and setting attributes ........................ 23
   3.4 The class of an object .................................. 23

4 Ordered and unordered factors ...................... 25
   4.1 A specific example ...................................... 25
   4.2 The function `tapply()` and ragged arrays ........... 26
   4.3 Ordered factors ......................................... 27
# Arrays and matrices

5.1 Arrays ................................................................................. 29
5.2 Array indexing. Subsections of an array ................................. 29
5.3 Index matrices ...................................................................... 30
5.4 The `array()` function ......................................................... 31
  5.4.1 Mixed vector and array arithmetic. The recycling rule .... 32
5.5 The outer product of two arrays ............................................ 32
5.6 Generalized transpose of an array ......................................... 33
5.7 Matrix facilities ................................................................. 33
  5.7.1 Matrix multiplication ....................................................... 34
  5.7.2 Linear equations and inversion ....................................... 34
  5.7.3 Eigenvalues and eigenvectors ......................................... 35
  5.7.4 Singular value decomposition and determinants .......... 35
  5.7.5 Least squares fitting and the QR decomposition .......... 36
5.8 Forming partitioned matrices, `cbind()` and `rbind()` ........ 37
5.9 The concatenation function, `c()`, with arrays ..................... 37
5.10 Frequency tables from factors ............................................. 38

# Lists and data frames

6.1 Lists ..................................................................................... 39
6.2 Constructing and modifying lists .......................................... 40
  6.2.1 Concatenating lists ......................................................... 40
6.3 Data frames .......................................................................... 40
  6.3.1 Making data frames ....................................................... 41
  6.3.2 `attach()` and `detach()` ............................................ 41
  6.3.3 Working with data frames ............................................. 42
  6.3.4 Attaching arbitrary lists ................................................ 42
  6.3.5 Managing the search path ............................................ 42

# Reading data from files

7.1 The `read.table()` function ................................................ 45
7.2 The `scan()` function ......................................................... 46
7.3 Accessing builtin datasets .................................................. 47
  7.3.1 Loading data from other R packages .............................. 47
7.4 Editing data .......................................................................... 47

# Probability distributions

8.1 R as a set of statistical tables ............................................... 49
8.2 Examining the distribution of a set of data .............................. 51
8.3 One- and two-sample tests .................................................. 55

# Grouping, loops and conditional execution

9.1 Grouped expressions .......................................................... 59
9.2 Control statements ............................................................. 59
  9.2.1 Conditional execution: `if` statements ......................... 59
  9.2.2 Repetitive execution: `for` loops, `repeat` and `while` .... 59
10 Writing your own functions

10.1 Simple examples ........................................... 61
10.2 Defining new binary operators ................................. 63
10.3 Named arguments and defaults ................................ 63
10.4 The ‘...’ argument ............................................ 64
10.5 Assignments within functions .................................. 64
10.6 More advanced examples ........................................ 65
  10.6.1 Efficiency factors in block designs ........................ 65
  10.6.2 Dropping all names in a printed array ...................... 65
  10.6.3 Recursive numerical integration ............................ 66
10.7 Scope ........................................................................ 67
10.8 Customizing the environment ..................................... 69
10.9 Classes, generic functions and object orientation .............. 70

11 Statistical models in R

11.1 Defining statistical models; formulae ............................. 73
  11.1.1 Contrasts ...................................................... 76
11.2 Linear models ..................................................... 76
11.3 Generic functions for extracting model information .......... 77
11.4 Analysis of variance and model comparison ..................... 78
  11.4.1 ANOVA tables ................................................. 78
11.5 Updating fitted models ............................................ 79
11.6 Generalized linear models ......................................... 79
  11.6.1 Families ........................................................ 80
  11.6.2 The glm() function .......................................... 81
11.7 Nonlinear least squares and maximum likelihood models ...... 84
  11.7.1 Least squares .................................................. 84
  11.7.2 Maximum likelihood .......................................... 86
11.8 Some non-standard models ......................................... 87

12 Graphical procedures

12.1 High-level plotting commands .................................... 89
  12.1.1 The plot() function ........................................... 90
  12.1.2 Displaying multivariate data ................................. 90
  12.1.3 Display graphics ................................................ 91
  12.1.4 Arguments to high-level plotting functions ................. 92
12.2 Low-level plotting commands ..................................... 93
  12.2.1 Mathematical annotation ....................................... 95
  12.2.2 Hershey vector fonts .......................................... 95
12.3 Interacting with graphics .......................................... 96
12.4 Using graphics parameters ........................................ 97
  12.4.1 Permanent changes: The par() function ..................... 97
  12.4.2 Temporary changes: Arguments to graphics functions ...... 98
12.5 Graphics parameters list .......................................... 98
  12.5.1 Graphical elements ............................................ 98
  12.5.2 Axes and tick marks .......................................... 100
Publisher’s Foreword

This manual introduces the use of R, an interactive environment for statistical computing.

R is free software. The term “free software” refers to your freedom to run, copy, distribute, study, change and improve the software. With R you have all these freedoms.

R is part of the GNU Project. The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. It was conceived as a way of bringing back the cooperative spirit that prevailed in the computing community in earlier days, by removing the obstacles to cooperation imposed by the owners of proprietary software.

You can support the GNU Project by becoming an associate member of the Free Software Foundation and paying regular membership dues. The Free Software Foundation is a tax-exempt charity dedicated to promoting the right to use, study, copy, modify, and redistribute computer programs. It also helps to spread awareness of the ethical and political issues of freedom in the use of software. For more information, visit the website www.fsf.org.

The development of R itself is guided by the R Foundation, a not for profit organization working in the public interest. Individuals and organizations using R can support its continued development by becoming members of the R Foundation. Further information is available at the website www.r-project.org.

Brian Gough
Publisher
May 2009
Authors’ Preface

This introduction to R is derived from an original set of notes describing the S and S-PLUS environments written by Bill Venables and David M. Smith (Insightful Corporation). We have made a number of small changes to reflect differences between the R and S programs, and expanded some of the material.

We would like to extend warm thanks to Bill Venables (and David Smith) for granting permission to distribute this modified version of the notes in this way, and for being a supporter of R from way back.

Comments and corrections are always welcome. Please address email correspondence to R-core@R-project.org.(1)

Suggestions to the reader

Most R novices will start with the introductory session in Appendix A. This should give some familiarity with the style of R sessions and more importantly some instant feedback on what actually happens.

Many users will come to R mainly for its graphical facilities. In this case, Chapter 12 [Graphics], page 89 on the graphics features can be read at almost any time and need not wait until all the preceding sections have been digested.

(1) Correspondence for this printed edition should be sent in the first instance to the publisher at info@network-theory.co.uk, as this version includes additional new material and examples. Comments will be passed onto the original authors as appropriate.
1 Introduction and preliminaries

1.1 The R environment

R is an integrated software environment for data manipulation, calculation and graphical display. Among other things it has

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either directly at the computer or on hardcopy, and
- a well developed, simple and effective programming language (called ‘S’) which includes conditionals, loops, user defined recursive functions and input and output facilities. (Indeed most of the system supplied functions are themselves written in this language.)

The term “environment” is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software.

1.2 Related software and documentation

R can be regarded as an implementation of the S language which was developed at Bell Laboratories by Rick Becker, John Chambers and Allan Wilks, and also forms the basis of the S-PLUS system.

The evolution of the S language is characterized by four books by John Chambers and coauthors. For R, the basic reference is The New S Language: A Programming Environment for Data Analysis and Graphics by Richard A. Becker, John M. Chambers and Allan R. Wilks. The new features of the 1991 release of S are covered in Statistical Models in S edited by John M. Chambers and Trevor J. Hastie. The formal methods and classes of the methods package are based on those described in Programming with Data by John M. Chambers. See Appendix D [References], page 127, for precise references.

There are now a number of books which describe how to use R for data analysis and statistics, and documentation for S/S-PLUS can typically be used with R, keeping the differences between the S implementations in mind. See section “What documentation exists for R?” in The R statistical system FAQ.\(^{(1)}\)

\(^{(1)}\) http://cran.r-project.org/faqs.html
1.3 R and statistics

Our introduction to the R environment did not mention statistics, yet many people use R as a statistics system. We prefer to think of it of an environment within which many classical and modern statistical techniques have been implemented. A few of these are built into the base R environment, but many are supplied as packages. There are about 25 packages supplied with R (called “standard” and “recommended” packages) and many more are available through the CRAN family of Internet sites (via http://CRAN.R-project.org) and elsewhere. More details on packages are given later (see Chapter 13 [Packages], page 107).

Most classical statistics and much of the latest methodology is available for use with R, but users may need to be prepared to do a little work to find it.

There is an important difference in philosophy between S (and hence R) and the other main statistical systems. In S, a statistical analysis is normally done as a series of steps, with intermediate results being stored in objects. Thus whereas SAS and SPSS will give copious output from a regression or discriminant analysis, R will give minimal output and store the results in a fit object for subsequent interrogation by further R functions.

1.4 R and the window system

The most convenient way to use R is at a graphics workstation running a windowing system. This guide is aimed at users who have this facility. In particular, we will occasionally refer to the use of R on an X window system, although the vast bulk of what is said applies generally to any implementation of the R environment.

Most users will find it necessary to interact directly with the operating system on their computer from time to time. In this guide, we mainly discuss interaction with the operating system on UNIX machines. If you are running R under Windows or Mac OS you will need to make some small adjustments.

Setting up a workstation to take full advantage of the customizable features of R is a straightforward procedure, and will not be considered further here. Users in difficulty can find further information at the R website www.r-project.org or seek local expert help.

1.5 Using R interactively

When you use the R program it issues a prompt when it expects input commands. The default prompt is “greater than” sign: >

which on UNIX might be the same as the shell prompt, and so it may appear that nothing is happening. However, as we shall see, it is easy to change to a different R prompt if you wish.see Section 10.8 [Customizing the environment], page 69 We will assume that the UNIX shell prompt is ‘$’.

In using R under UNIX, the suggested procedure for the first run of the program is as follows:
1. Create a separate sub-directory, say `work`, to hold data files on which you will use R for this problem, and enter the directory. This will be the working directory whenever you use R for this particular problem.

   ```
   $ mkdir work
   $ cd work
   ```

2. Start the R program with the command

   ```
   $ R
   R version 2.9.0
   >
   ```

3. At this point R commands may be issued (see later).

4. To quit the R program, the command is

   ```
   > q()
   ```

   You will be asked whether you want to save the data from your R session. On some systems this will bring up a dialog box, and on others you will receive a text prompt to which you can respond `yes`, `no` or `cancel` (a single letter abbreviation will do) to save the data before quitting, quit without saving, or return to the R session. Data which is saved will be available in future R sessions.

Further R sessions are simple.

1. Change to the directory `work` and start the program as before:

   ```
   $ cd work
   $ R
   R version 2.9.0
   >
   ```

2. Use the R program, terminating with the `q()` command at the end of the session.

To use R under Windows, the procedure is basically the same. Create a folder as the working directory, and set the `Start In` field in your R shortcut to the folder path. Then launch R by double-clicking on the icon.

### 1.6 An introductory session

Readers wishing to get a feel for R at a computer before proceeding are strongly advised to work through the introductory session given in Appendix A [A sample session], page 111.
1.7 Getting help with functions and features

R has a built-in help facility similar to the man command of UNIX. To get more information on any specific named function, for example solve, the command is

\>
help(solve)

An alternative is

\>
?solve

On Unix systems, the help output is displayed using the default pager. Hit the (SPACE) key to scroll forwards through the text, or [q] to quit and return to the R prompt. For a feature specified by special characters, the argument must be enclosed in double or single quotes, making it a “character string”:

\>
help("[[")

This is also necessary for words with syntactic meaning, including if, for and function. If in doubt, always use quotes. Either form of quote mark may be used to escape the other, as in the string "It's important". Our convention is to use double quote marks for preference.

The help.search command (alternatively ??) allows searching for help in various ways. For example,

\>
??solve

Help files with alias or concept or title matching ‘solve’ using fuzzy matching:

base::backsolve Solve an Upper or Lower Triangular System
base::qr The QR Decomposition of a Matrix
base::solve Solve a System of Equations

......

The name of each entry is given on the left-hand side, followed by a brief description. Entries can be displayed with a command like ?base::qr, which shows the help page for the qr function in the base package.

It is also possible to search for longer strings:

\>
??"linear models"

Help files with alias or concept or title matching 'linear models' using fuzzy matching:

MASS::boxcox Box-Cox Transformations for Linear Models
MASS::glm.nb Fit a Negative Binomial Generalized Linear Model
MASS::lm.gls Fit Linear Models by Generalized Least Squares

......

Try ?help.search for details and more examples.

The examples on a help topic can normally be run by

\>
example(topic)

On most R installations help is available in HTML format by running
which will launch a Web browser that allows the help pages to be browsed with hyperlinks. On UNIX, subsequent help requests are sent to the HTML-based help system. The 'Search Engine and Keywords' link in the page loaded by `help.start()` is particularly useful as it is contains a high-level concept list which searches through the available functions. It can be a great way to get your bearings quickly and to understand the breadth of what R has to offer.

Windows versions of R have other optional help systems: use

```r
> ?help
```

for further details.

### 1.8 R commands, case sensitivity, etc.

Technically, R is an *expression language* with a very simple syntax. It is *case sensitive*, as are most UNIX-based packages, so `A` and `a` are different symbols and refer to different variables. The set of symbols which can be used in R function and variable names depends on the operating system and its language settings (technically on the *locale* in use). Normally all alphanumeric symbols are allowed\(^{(2)}\) (and in some locales this includes accented letters) plus `.` and `_`, with the restriction that a name must start with `.` or a letter, and if it starts with `.` the second character must not be a digit.

Elementary commands consist of either *expressions* or *assignments*. If an expression is given as a command, it is evaluated, printed (unless specifically made invisible), and the value is discarded. An assignment evaluates an expression and passes the value to a variable, but the result is not automatically printed.

Commands are separated either by a semi-colon (`;`), or by a newline. Elementary commands can be grouped together into one compound expression by braces (`{` and `}`). *Comments* can be put almost\(^{(3)}\) anywhere, starting with a hashmark (`#`), everything to the end of the line is a comment.

If a command is not complete at the end of a line, R will give a different prompt, by default

```
+
```

on second and subsequent lines, and continue to read input until the command is syntactically complete. This prompt may be changed by the user. In this book, we will generally omit the continuation prompt and indicate continuation by simple indenting.

Command lines entered at the console are limited\(^{(4)}\) to about 4095 bytes (not characters).

---

\(^{(2)}\) For portable R code (including that to be used in R packages) only A–Za–z0–9 should be used.

\(^{(3)}\) *not* inside strings, nor within the argument list of a function definition

\(^{(4)}\) some of the consoles will not allow you to enter more, and amongst those which do some will silently discard the excess and some will use it as the start of the next line.
The result of a command is printed to the current output device. If the result is an array, such as a vector or matrix, then the elements are formatted with line breaks (if necessary) and the indices of the leading entries are labelled in square brackets, [index]. For example, an array of 20 elements might be displayed as follows,

```r
> array(0,20)
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [15] 0 0 0 0 0
```

The labels ‘[1]’ and ‘[15]’ indicate the first and fifteenth elements in the output. It is important to note that these labels are not part of the data itself.

The index labels for matrices are placed at the start of each row and column in the output:

```r
> matrix(0,4,4)
[1,] 0 0 0 0
[2,] 0 0 0 0
[3,] 0 0 0 0
[4,] 0 0 0 0
```

For simplicity, index labels are usually omitted in the examples in this book.

### 1.9 Recall and correction of previous commands

Under many versions of UNIX and on Windows, R provides a mechanism for recalling and re-executing previous commands. The vertical arrow keys on the keyboard can be used to scroll forward and backward through a command history. Once a command is located in this way, the cursor can be moved within the command using the horizontal arrow keys, and characters can be deleted with the **DEL** key or inserted with the other keys. More details are provided later: see Appendix C [The command-line editor], page 123.

The recall and editing capabilities under UNIX are highly customizable. You can find out how to do this by reading the documentation entry for the GNU readline library, using `man readline` or `info readline` at the Unix prompt (if installed).

Alternatively, the GNU Emacs text editor provides more general support mechanisms (via ESS, the *Emacs Speaks Statistics* mode) for working interactively with R. See the section “R and Emacs” in the R FAQ for more information.
1.10 Executing commands from or diverting output to a file

If commands\(^{(5)}\) are stored in an external file, such as `commands.R` in the working directory ‘work’, they may be executed at any time in an R session with the command

```r
> source("commands.R")
```

For Windows, **Source** is also available on the **File** menu. The function **sink**,

```r
> sink("record.lis")
```

will divert all subsequent output from the console to an external file, ‘record.lis’. The command

```r
> sink()
```
restores output to the console once again.

1.11 Data permanency and removing objects

The entities that R creates and manipulates are known as *objects*. These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components.

During an R session, objects are created and stored by name (we discuss this process in the next session). The command

```r
> objects()
```

can be used to display the names of (most of) the objects which are currently stored within R. The same information can also be displayed with the `ls()` command. The collection of objects currently stored is called the *workspace*.

The function **rm** removes objects from the workspace:

```r
> rm(x, y, z, ink, junk, temp, foo, bar)
```

All objects created during an R session can be stored permanently in a file for use in future R sessions. At the end of each R session you are given the opportunity to save all the currently available objects. If you indicate that you want to do this, the objects are written to a file called `.RData`\(^{(6)}\) in the current directory, and the command lines used in the session are saved to a file called `.Rhistory`.

When R is started at a later time from the same directory it reloads the workspace from this file. At the same time the associated command history is reloaded.

It is recommended that you should use separate working directories for each analysis conducted with R. It is quite common for objects to be created with names such as `x` and `y`, which are meaningful in the context of a single analysis, but difficult to identify if several analyses have been carried out in the same directory.

---

\(^{(5)}\) of unlimited length.  
\(^{(6)}\) The leading “dot” in this file name makes it *invisible* in normal file listings in UNIX.
2 Simple manipulations; numbers and vectors

2.1 Vectors and assignment

R operates on named data structures. The simplest such structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers. To set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command

\[ x \leftarrow \text{c}(10.4, 5.6, 3.1, 6.4, 21.7) \]

This is an assignment statement using the function c() which in this context can take an arbitrary number of vector arguments and whose value is a vector obtained by concatenating the arguments end to end. (1)

A number occurring by itself in an expression is taken as a vector of length one.

Note that the assignment operator ‘\(\leftarrow\)’ consists of the two characters ‘<’ (“less than”) and ‘-’ (“minus”) occurring strictly side-by-side and it ‘points’ to the object receiving the value of the expression.

Assignments can also be made using the function assign(). An equivalent way of making the assignment shown above is

\[ \text{assign(“x”, c(10.4, 5.6, 3.1, 6.4, 21.7))} \]

The operator \(\leftarrow\) can be thought of as a syntactic short-cut for the assign command.

Assignments can also be made in the other direction, using ‘\(-\)’ as an alternative assignment operator. So the same assignment could be made using

\[ \text{c}(10.4, 5.6, 3.1, 6.4, 21.7) \rightarrow x \]

If an expression is used as a complete command, the value is printed and discarded (2). So now if we were to use the command

\[ 1/x \]

the reciprocals of the five values above would be printed at the terminal (and the value of \(x\) would be unchanged).

The further assignment

\[ y \leftarrow \text{c}(x, 0, x) \]

would create a vector \(y\) with 11 entries, consisting of two copies of \(x\) with a zero in the middle place.

---

(1) With other than vector types of argument, such as list mode arguments, the action of c() is rather different. See Section 6.2.1 [Concatenating lists], page 40.

(2) Actually, the previous value is still available as .Last.value before any other statements are executed.
2.2 Vector arithmetic

Vectors can be used in arithmetic expressions, in which case the operations are performed element-by-element. Vectors occurring in the same expression need not all be of the same length. If they are not, the resulting value of the expression is a vector with the same length as the longest vector which occurs in the expression. Shorter vectors in the expression are recycled as often as need be (perhaps fractionally) until they match the length of the longest vector. In particular, a constant is simply repeated. So with the above assignments the command

\[
> v <- 2*x + y + 1
\]
generates a new vector \( v \) of length 11 constructed by adding together, element-by-element, \( 2\times x \) repeated 2.2 times, \( y \) repeated just once, and 1 repeated 11 times.

The elementary arithmetic operators use the normal symbols \( +, -, *, / \), and the exponentiation operator \( ^ \) for raising to a power. In addition, all of the common mathematical functions are available. \( \log, \exp, \sin, \cos, \tan, \sqrt{\text{ }} \), and so on, with their usual meaning. The functions \( \text{max} \) and \( \text{min} \) select the largest and smallest elements of a vector respectively. \( \text{range} \) is a function whose value is a vector of length two, namely \( c(\text{min}(x), \text{max}(x)) \). The function \( \text{length}(x) \) is the number of elements in \( x \), \( \text{sum}(x) \) gives the total of the elements in \( x \), and \( \text{prod}(x) \) their product.

Two statistical functions are \( \text{mean}(x) \) which calculates the sample mean, \( \frac{x}{\text{length}(x)} \) and \( \text{var}(x) \) which gives the sample variance

\[
\frac{\text{sum}((x-\text{mean}(x))^2)}{\text{length}(x)-1}
\]
or sample variance. If the argument to \( \text{var()} \) is an \( n \)-by-\( p \) matrix the value is a \( p \)-by-\( p \) sample covariance matrix found by regarding the rows as independent \( p \)-variate sample vectors.

\( \text{sort}(x) \) returns a vector of the same size as \( x \) with the elements arranged in increasing order; however there are other more flexible sorting facilities available, such as \( \text{order()} \) or \( \text{sort.list()} \) which produce a permutation to do the sorting.

Note that \( \text{max} \) and \( \text{min} \) select the largest and smallest values in their arguments, even if they are given several vectors. The \( \text{parallel} \) maximum and minimum functions \( \text{pmax} \) and \( \text{pmin} \) return a vector (of length equal to their longest argument) that contains in each element the maximum or minimum element in that position in any of the input vectors.

For most purposes, the user will not be concerned if the “numbers” in a numeric vector are integers, reals or even complex. Internally, calculations are done as double precision real numbers, or double precision complex numbers if the input data are complex.

To work with complex numbers, supply an explicit complex part. Thus, the expression

\[
\text{sqrt}(-17)
\]
will give NaN (Not a Number) and a warning, but
\[ \sqrt{-17 + 0i} \]
will do the computations as complex numbers.

### 2.3 Generating regular sequences

R has a number of commands for generating sequences of numbers. The most important sequence operator is the colon `:` which produces a linear range of values. For example, `1:30` is the vector
\[ c(1, 2, 3, \ldots, 29, 30) \]
The colon operator has high priority within an expression, so, for example `2*1:15` is the vector `c(2, 4, \ldots, 28, 30)`. Put `n <- 10` and compare the sequences `1:n-1` and `1:(n-1)` to see the difference.

The construction `n:1` may be used to generate a sequence backwards.

The function `seq()` is a more general function for generating sequences. It has five arguments, only some of which may be specified in any one call. The first two arguments, if given, specify the beginning and end of the sequence. If these are the only two arguments given the result is the same as the colon operator, i.e. `seq(2, 10)` is equivalent to `2:10`.

Parameters to `seq()`, and to many other R functions, can also be given in named form, in which case the order in which they appear is irrelevant. The first two parameters may be named `from=value` and `to=value`; thus `seq(1, 30)`, `seq(from=1, to=30)` and `seq(to=30, from=1)` are all the same as `1:30`. The next two parameters to `seq()` may be named `by=value` and `length=value`, which specify a step size and a length for the sequence respectively. If neither of these is given, the default `by=1` is assumed.

For example,
\[
> \text{seq}(-5, 5, \text{by}=.2) \rightarrow s3
\]
generates the vector `c(-5.0, -4.8, -4.6, \ldots, 4.6, 4.8, 5.0)` and stores it in `s3`. Similarly,
\[
> \text{s4} <- \text{seq(length}=51, \text{from}=-5, \text{by}=.2)
\]
generates the same vector in `s4`.

The fifth parameter may be named `along=vector`. If used, `along` must be the only parameter, and creates a sequence `1, 2, \ldots, \text{length}(vector)`, or the empty sequence if the vector is empty (as it can be).

A related function is `rep()` which replicates objects in various ways. The simplest form is
\[
> \text{s5} <- \text{rep(x, times}=5)
\]
which will put five copies of `x` end-to-end in `s5`. Another useful version is
\[
> \text{s6} <- \text{rep(x, each}=5)
\]
which repeats each element of `x` five times before moving on to the next.
2.4 Logical vectors

As well as numerical vectors, R also allows manipulation of logical quantities. The elements of a logical vector can have the values `TRUE`, `FALSE`, and `NA` (for “not available”, see below). The first two are often abbreviated as `T` and `F`, respectively. Note however that `T` and `F` are just variables which are set to `TRUE` and `FALSE` by default, but are not reserved words and can be overwritten by the user. Hence, you should always use `TRUE` and `FALSE`.

Logical vectors are generated by conditions. For example,

```r
> temp <- x > 13
```

sets `temp` as a vector of the same length as `x` with values `FALSE` corresponding to elements of `x` where the condition is not met and `TRUE` where it is.

The logical operators are `<`, `<=`, `>`, `>=`, `==` for exact equality and `!=` for inequality. In addition, if `c1` and `c2` are logical expressions, then `c1 & c2` is their intersection (“and”), `c1 | c2` is their union (“or”), and `!c1` is the negation of `c1`.

Logical vectors may be used in ordinary arithmetic, where they are coerced into numeric vectors, `FALSE` becoming `0` and `TRUE` becoming `1`. However, there are situations where logical vectors and their coerced numeric counterparts are not equivalent, as in the following examples.

2.5 Missing values

In some cases, the components of a vector may not be completely known. When an element or value is “not available” or a “missing value” in the statistical sense, a place within a vector may be reserved for it by assigning it the special value `NA`. In general, any operation on an `NA` becomes an `NA`. The motivation for this rule is simply that if the specification of an operation is incomplete the result cannot be known, and hence is not available.

The function `is.na(x)` gives a logical vector of the same size as `x` with value `TRUE` if and only if the corresponding element in `x` is `NA`.

```r
> z <- c(1:3,NA); ind <- is.na(z)
```

Notice that the logical expression `x == NA` is quite different from `is.na(x)` since `NA` is not really a value but a marker for a quantity that is not available. Thus `x == NA` is a vector of the same length as `x` all of whose values are `NA` as the logical expression itself is incomplete and hence undecidable.

Note that there is a second kind of “missing” value produced by numerical computation, the so-called `Not a Number`, `NaN`, values. Examples are

```r
> 0/0
```

or

```r
> Inf - Inf
```

which both give `NaN` since the result cannot be defined sensibly. The presence of `NaN` values can be tested using the function `is.nan(x)`.

In summary, `is.na(x)` is `TRUE` both for `NA` and `NaN` values. To differentiate these, `is.nan(x)` is only `TRUE` for `NaNs`. 
Chapter 2: Simple manipulations; numbers and vectors

Missing values are sometimes printed as \texttt{<NA>} when character vectors are printed without quotes.

### 2.6 Character vectors

Character quantities and character vectors are used frequently in R, for example as plot labels and titles. They are denoted by a sequence of characters delimited by the double quote character, e.g., \texttt{"x-values"}, \texttt{"New iteration results"}.

Character strings can be entered using either matching double (\texttt{"\texttt{\textbackslash}"}) or single (\texttt{'\texttt{\textbackslash}\texttt{'}) quotes, but are printed using double quotes (or sometimes without quotes). They use C-style escape sequences, with \texttt{\textbackslash} as the escape character. The backslash character itself is entered and printed as \texttt{\textbackslash\textbackslash}, and inside double quotes \texttt{"\textbackslash} is entered as \texttt{\textbackslash	exttt{"}}. Other useful escape sequences are \texttt{\textbackslash n}, newline, \texttt{\textbackslash t}, tab and \texttt{\textbackslash b}, backspace—see \texttt{\?Quotes} for a full list.

Character vectors may be concatenated into a vector by the \texttt{c()} function.

The \texttt{paste()} function takes an arbitrary number of arguments and concatenates them one by one into character strings. Any numbers given among the arguments are coerced into character strings in the same way they would be if they were printed. By default, the arguments are separated by a single blank character in the result, but this can be controlled with the named parameter \texttt{sep=string}, which changes the separator to \texttt{string} (which may also be an empty string).

For example,

\begin{verbatim}
> labs <- paste(c("X","Y"), 1:10, sep="")
\end{verbatim}

makes \texttt{labs} into the character vector

\begin{verbatim}
c("X1", "Y2", "X3", "Y4", "X5", "Y6", "X7", "Y8", "X9", "Y10")
\end{verbatim}

Note that recycling of short lists takes place here too; thus \texttt{c("X", "Y")} is repeated 5 times to match the sequence \texttt{1:10}.

(3) \texttt{paste(\ldots, collapse=ss)} joins the arguments into a single character string putting \texttt{ss} in between. There are more tools for character manipulation, see the help for \texttt{sub} and \texttt{substring}. 

2.7 Index vectors; selecting and modifying subsets of a data set

Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets, \textit{v[i]}. More generally, any expression that evaluates to a vector may have subsets of its elements similarly selected by appending an index vector in square brackets immediately after the expression, \textit{(expression)[i]}.

Such index vectors can be any of four distinct types.

1. A \textbf{logical vector}. In this case the index vector must be of the same length as the vector from which elements are to be selected. Values corresponding to \texttt{TRUE} in the index vector are selected and those corresponding to \texttt{FALSE} are omitted. For example,

\begin{verbatim}
> y <- x[!is.na(x)]
\end{verbatim}

creates (or re-creates) an object \texttt{y} which will contain the non-missing values of \texttt{x}, in the same order. Note that if \texttt{x} has missing values, \texttt{y} will be shorter than \texttt{x}. Also

\begin{verbatim}
> (x+1)[(!is.na(x)) & x>0] -> z
\end{verbatim}

creates an object \texttt{z} and places in it the values of the vector \texttt{x+1} for which the corresponding value in \texttt{x} was both non-missing and positive.

2. A \textbf{vector of positive integral quantities}. In this case the values in the index vector must lie in the set \{1, 2, \ldots, length(x)\}. The corresponding elements of the vector are selected and concatenated, \textit{in that order}, in the result. The index vector can be of any length and the result is of the same length as the index vector. For example, \texttt{x[6]} is the sixth component of \texttt{x} and

\begin{verbatim}
> x[1:10]
\end{verbatim}

selects the first 10 elements of \texttt{x} (assuming \texttt{length(x)} is not less than 10). Also

\begin{verbatim}
> c("x","y")[rep(c(1,2,2,1), times=4)]
\end{verbatim}

(an admittedly unlikely thing to do) produces a character vector of length 16 consisting of "x", "y", "y", "x" repeated four times.

3. A \textbf{vector of negative integral quantities}. Such an index vector specifies the values to be \textit{excluded} rather than included. Thus

\begin{verbatim}
> y <- x[-(1:5)]
\end{verbatim}

gives \texttt{y} all but the first five elements of \texttt{x}.

4. A \textbf{vector of character strings}. This possibility only applies where an object has a \texttt{names} attribute to identify its components. In this case a sub-vector of the names vector may be used in the same way as the positive integral labels in item 2 above.
Index

! .............................................. 16
!=.............................................. 16
%0 ............................................. 34
%*%............................................. 32
&.............................................. 16
&.............................................. 16
&&........................................... 59
* ............................................. 14
*............................................. 14
+ ............................................. 14
+ ............................................. 14
- ............................................. 14
-> ............................................ 13
. ............................................. 79
.First ...................................... 70
.Last ....................................... 70
.Last.value ................................ 13
.Rdata ...................................... 11
.Rprofile ................................... 69
/
/............................................. 14
:
:............................................. 15
::.......................................... 110
:::.......................................... 110
<........................................... 16
<........................................... 16
<=......................................... 13
<=......................................... 68
<=......................................... 16
=........................................... 16
==......................................... 16
>........................................... 16
>........................................... 16
?............................................ 8
??.......................................... 8
~............................................ 14
|........................................... 16
||.......................................... 59
~ ............................................ 73
A
Abbreviating component names ........ 39
abline...................................... 93
Accessing builtin datasets .......... 47
ace.......................................... 87
add1 ....................................... 79
Additive models ....................... 87
Allowed characters ................... 9
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis of variance (ANOVA)</td>
<td>78</td>
</tr>
<tr>
<td>Annotation, of plots</td>
<td>95</td>
</tr>
<tr>
<td>anova</td>
<td>77, 78</td>
</tr>
<tr>
<td>ANOVA tables</td>
<td>78</td>
</tr>
<tr>
<td>aov</td>
<td>78</td>
</tr>
<tr>
<td>aperm</td>
<td>33</td>
</tr>
<tr>
<td>Arithmetic functions and operators</td>
<td>14</td>
</tr>
<tr>
<td>array</td>
<td>31</td>
</tr>
<tr>
<td>Arrays</td>
<td>29</td>
</tr>
<tr>
<td>as.data.frame</td>
<td>41</td>
</tr>
<tr>
<td>as.vector</td>
<td>37</td>
</tr>
<tr>
<td>assign</td>
<td>13</td>
</tr>
<tr>
<td>Assignment</td>
<td>13</td>
</tr>
<tr>
<td>Assignment, compared with equality</td>
<td>13</td>
</tr>
<tr>
<td>Assignment, in reverse direction</td>
<td>13</td>
</tr>
<tr>
<td>Assignment, of vectors</td>
<td>13</td>
</tr>
<tr>
<td>Assignment, within functions</td>
<td>64</td>
</tr>
<tr>
<td>attach</td>
<td>41</td>
</tr>
<tr>
<td>attr</td>
<td>23</td>
</tr>
<tr>
<td>attributes</td>
<td>23</td>
</tr>
<tr>
<td>Attributes</td>
<td>21</td>
</tr>
<tr>
<td>avas</td>
<td>87</td>
</tr>
<tr>
<td>Axes, in plots</td>
<td>100</td>
</tr>
<tr>
<td>axis</td>
<td>94</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Base package</td>
<td>107</td>
</tr>
<tr>
<td>Beta distribution</td>
<td>49</td>
</tr>
<tr>
<td>Binary operators</td>
<td>63</td>
</tr>
<tr>
<td>Binomial distribution</td>
<td>49</td>
</tr>
<tr>
<td>Block design, example</td>
<td>30</td>
</tr>
<tr>
<td>Block designs, efficiency factors</td>
<td>65</td>
</tr>
<tr>
<td>bootstrap methods</td>
<td>108</td>
</tr>
<tr>
<td>Box plots</td>
<td>55</td>
</tr>
<tr>
<td>boxplot</td>
<td>55</td>
</tr>
<tr>
<td>break</td>
<td>60</td>
</tr>
<tr>
<td>bruto</td>
<td>87</td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>13, 17, 37, 40</td>
</tr>
<tr>
<td>coefficients</td>
<td></td>
</tr>
<tr>
<td>Chi-squared distribution</td>
<td>49</td>
</tr>
<tr>
<td>Classes</td>
<td>23, 70</td>
</tr>
<tr>
<td>classification functions</td>
<td>108</td>
</tr>
<tr>
<td>cluster analysis</td>
<td>108</td>
</tr>
<tr>
<td>Coercion</td>
<td>22</td>
</tr>
<tr>
<td>Color, in plots</td>
<td>99</td>
</tr>
<tr>
<td>Command-line editing</td>
<td>123</td>
</tr>
<tr>
<td>command-line options</td>
<td>115</td>
</tr>
<tr>
<td>Commands, separating</td>
<td>9</td>
</tr>
<tr>
<td>Comment character</td>
<td>9</td>
</tr>
<tr>
<td>Complex numbers, using</td>
<td>14</td>
</tr>
<tr>
<td>Component names, abbreviating</td>
<td>39</td>
</tr>
<tr>
<td>Concatenating lists</td>
<td>40</td>
</tr>
<tr>
<td>Concatenation of arrays</td>
<td>37</td>
</tr>
<tr>
<td>Continuation lines</td>
<td>9</td>
</tr>
<tr>
<td>contour</td>
<td>91</td>
</tr>
<tr>
<td>contrasts</td>
<td>76</td>
</tr>
<tr>
<td>Contributed packages</td>
<td>108</td>
</tr>
<tr>
<td>Control statements</td>
<td>59</td>
</tr>
<tr>
<td>Conversion, between numbers and characters</td>
<td>22</td>
</tr>
<tr>
<td>Covariance matrix, computed using var</td>
<td>14</td>
</tr>
<tr>
<td>CRAN, package archive</td>
<td>6, 108</td>
</tr>
<tr>
<td>crossprod</td>
<td>31, 34</td>
</tr>
<tr>
<td>Cumulative distribution function</td>
<td>49</td>
</tr>
<tr>
<td>Cursor keys</td>
<td>124</td>
</tr>
<tr>
<td>Customizing the environment</td>
<td>69</td>
</tr>
<tr>
<td>cut</td>
<td>38</td>
</tr>
<tr>
<td>Cyrillic characters, in plots</td>
<td>95</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>Dashed lines, in plots</td>
<td>99</td>
</tr>
<tr>
<td>data</td>
<td>47</td>
</tr>
<tr>
<td>Data frames</td>
<td>40</td>
</tr>
<tr>
<td>datasets, builtin</td>
<td>41</td>
</tr>
<tr>
<td>Debugging, with the R debugger</td>
<td>118</td>
</tr>
<tr>
<td>Default values</td>
<td>63</td>
</tr>
<tr>
<td>Deleting objects</td>
<td>11</td>
</tr>
<tr>
<td>density</td>
<td>52</td>
</tr>
<tr>
<td>density estimation</td>
<td>108</td>
</tr>
<tr>
<td>Density estimation</td>
<td>52</td>
</tr>
<tr>
<td>------</td>
<td>---</td>
</tr>
<tr>
<td>Density function</td>
<td>49</td>
</tr>
<tr>
<td>det</td>
<td>36</td>
</tr>
<tr>
<td>detach</td>
<td>41</td>
</tr>
<tr>
<td>determinant</td>
<td>36</td>
</tr>
<tr>
<td>Determinants</td>
<td>36</td>
</tr>
<tr>
<td>dev.list</td>
<td>105</td>
</tr>
<tr>
<td>dev.next</td>
<td>105</td>
</tr>
<tr>
<td>dev.off</td>
<td>106</td>
</tr>
<tr>
<td>dev.prev</td>
<td>105</td>
</tr>
<tr>
<td>dev.set</td>
<td>105</td>
</tr>
<tr>
<td>deviance</td>
<td>77</td>
</tr>
<tr>
<td>Devices, for graphics</td>
<td>104</td>
</tr>
<tr>
<td>diag</td>
<td>34</td>
</tr>
<tr>
<td>dim</td>
<td>29</td>
</tr>
<tr>
<td>dimension vector</td>
<td>27</td>
</tr>
<tr>
<td>Distributions</td>
<td>49</td>
</tr>
<tr>
<td>Diverting input and output</td>
<td>11</td>
</tr>
<tr>
<td>dotchart</td>
<td>91</td>
</tr>
<tr>
<td>Dotted lines, in plots</td>
<td>99</td>
</tr>
<tr>
<td>drop1</td>
<td>79</td>
</tr>
<tr>
<td>Dropping names, from a printed array</td>
<td>65</td>
</tr>
<tr>
<td>Dynamic graphics</td>
<td>106</td>
</tr>
<tr>
<td>Extracting model information</td>
<td>77</td>
</tr>
<tr>
<td><strong>F</strong></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>16</td>
</tr>
<tr>
<td>F distribution</td>
<td>49</td>
</tr>
<tr>
<td>factor</td>
<td>25</td>
</tr>
<tr>
<td>Factors</td>
<td>25, 76</td>
</tr>
<tr>
<td>FALSE</td>
<td>16</td>
</tr>
<tr>
<td>Families</td>
<td>80</td>
</tr>
<tr>
<td>Figure margins</td>
<td>101</td>
</tr>
<tr>
<td>Fitting, least squares</td>
<td>36</td>
</tr>
<tr>
<td>fivenum</td>
<td>51</td>
</tr>
<tr>
<td>Fonts, in graphics</td>
<td>95</td>
</tr>
<tr>
<td>Fonts, in plots</td>
<td>99</td>
</tr>
<tr>
<td>for</td>
<td>59</td>
</tr>
<tr>
<td>formal methods</td>
<td>109</td>
</tr>
<tr>
<td>formula</td>
<td>77</td>
</tr>
<tr>
<td>Formulae, for statistical models</td>
<td>73</td>
</tr>
<tr>
<td>Frequency tables</td>
<td>38</td>
</tr>
<tr>
<td>function</td>
<td>61</td>
</tr>
<tr>
<td>Functions, recursive</td>
<td>66</td>
</tr>
<tr>
<td>Functions, writing your own</td>
<td>61</td>
</tr>
<tr>
<td><strong>G</strong></td>
<td></td>
</tr>
<tr>
<td>Gamma distribution</td>
<td>49</td>
</tr>
<tr>
<td>generalized additive models</td>
<td>109</td>
</tr>
<tr>
<td>Generalized linear models</td>
<td>79</td>
</tr>
<tr>
<td>Generalized transpose of an array</td>
<td>33</td>
</tr>
<tr>
<td>Generic functions</td>
<td>70</td>
</tr>
<tr>
<td>Generic functions, for extracting model information</td>
<td>77</td>
</tr>
<tr>
<td>Geometric distribution</td>
<td>49</td>
</tr>
<tr>
<td>getAnywhere</td>
<td>71</td>
</tr>
<tr>
<td>getS3method</td>
<td>71</td>
</tr>
<tr>
<td>glm</td>
<td>81</td>
</tr>
<tr>
<td>graphics</td>
<td>108</td>
</tr>
<tr>
<td>Graphics</td>
<td>89</td>
</tr>
<tr>
<td>Graphics device drivers</td>
<td>104</td>
</tr>
<tr>
<td>Graphics parameters</td>
<td>97</td>
</tr>
<tr>
<td>grid graphics</td>
<td>108</td>
</tr>
<tr>
<td>Grouped expressions</td>
<td>59</td>
</tr>
<tr>
<td><strong>H</strong></td>
<td></td>
</tr>
<tr>
<td>hazard function</td>
<td>49</td>
</tr>
<tr>
<td>help</td>
<td>8</td>
</tr>
<tr>
<td>help.search</td>
<td>8</td>
</tr>
</tbody>
</table>
recursive partitioning 109
Recycling rule 14, 32
References 127
regression trees 109
Regression, least squares 36
Regular sequences 15
Removing objects 11
rep 15
repeat 60
resid 77
residuals 77
RHOME 115
rlm 87
rm 11
Robust regression 87
RProfile.site 69

S
Sample session 111
Sample variance, computed using var 14
SAS, importing data from 108
saving an R session 7
scan 46
Scope 67
sd 26
search 42, 107
Search path 42
Selecting elements of a vector 18
selecting points, with the mouse 96
sep 17
Separating commands 9
Separator, for output 17
seq 15
Sequence, making by repetition 15
Sequences, generating 15
Shapiro-Wilk test 54
shapiro.test 54
sin 14
Singular value decomposition 35
sink 11
solve 34
sort 14
source 11
sparse matrices 109
spatial statistics 109
splines 109
split 60
SPSS, importing data from 108
sqrt 14
Standard (base) packages 107
Start-up files 69
starting R 6
starting R, under Windows 7
Stata, importing data from 108
Statistical models 73
stem 51
step 77, 79
Storing output to a file 11
Strings, used as vector index 18
Student's t distribution 49
Student's t test 56
Subsets, of a vector 18
sum 14
summary 51, 78
survival analysis 109
svd 35

T
t 33
T 16
t.test 56
table 31, 38
Tabulation 38
tails, of distribution 49
tan 14
tapply 26
Tcl/Tk 109
text 93
Tick marks, in plots 100
title 94
TMPDIR 115
tree 87
Tree-based models 87
trellis graphics 108
TRUE 16
Truncating objects 22
Two-way table of frequencies 38
Type, of object components 21

U
unclass 23
Uniform distribution 49
Unknown values (NA) 16
update 79
Index

Updating fitted models ............... 79

V

var ................................. 14, 26
var.test ............................. 57
Variance, analysis of (ANOVA) ...... 78
vcov ................................. 78
vector ............................... 13
Vector, of length one .................. 13
Vectors ................................ 13
Vectors, arithmetic on ................. 14
Vectors, combining different lengths .. 14
Vectors, indexing ........................ 18

W

Weibull distribution .................... 49
while .................................... 60
Width, of lines in plots ............... 99
wilcox.test ............................ 57
Wilcoxon distribution .................. 49
Wilcoxon test ........................... 57
windows ................................ 104
Windows, starting R .................... 120
Workspace ................................ 11
Workspace, listing objects in .......... 11
Writing functions ....................... 61
Writing output to a file ................ 11

X

X11 ................................... 104