

'R' Complements to

# Modern Applied Statistics with S-Plus

Second edition

by

W. N. Venables and B. D. Ripley  
Springer (1997). ISBN 0-387-98214-0

12 Aug 1999

These complements have been produced to supplement the second edition of MASS. This is the final version of these complements as the third edition is now out. The definitive source is <http://www.stats.ox.ac.uk/pub/MASS2/>.

© W. N. Venables and B. D. Ripley 1998–9. A licence is granted for personal study and classroom use. Redistribution in any other form is prohibited.

Selectable links are [in this colour](#).  
Selectable URLs are [in this colour](#).

# Introduction

These complements are made available on-line to supplement the book for users of the package R. The general convention is that material here should be thought of as following the material in the chapter in the book.

The aim of these complements is to make the book usable to users who only have access to R, and also to help experienced S-PLUS users make use of R.

We are grateful to the R developers and especially Kurt Hornik for their comments and their efforts in making R 0.62.2 and later compatible with the S code we use.

There are separate Complements documents for S-PLUS 4.x, 5.x and for S programming and statistical methods available from

<http://www.stats.ox.ac.uk/pub/MASS2/>.

Most of these are of little interest to R users; perhaps most relevant are Sections 4.12, 4.13 and 4.16 of the programming complements, as well as Sections 5.5, 5.6, 7.5, 11.1, 11.2, 11.4, 12.1, 13.3, 14.4, 17.3 and 17.4 of the statistics complements which describe methods in packages by ourselves and others which are available for R.

# Contents

<b>Introduction to Complements</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 A quick overview of R . . . . .	2
1.2 Using R under Unix . . . . .	3
1.3 Using R under Windows . . . . .	6
<b>2 The S Language</b>	<b>7</b>
2.1 Differences between R and S . . . . .	7
2.11 Customizing your R environment . . . . .	10
2.13 BATCH operation . . . . .	11
<b>3 Graphical Output</b>	<b>12</b>
3.3 Enhancing plots . . . . .	12
3.7 The R colour model . . . . .	13
<b>4 Programming in S</b>	<b>15</b>
4.5 Editing, correcting and documenting functions . . . . .	15
4.6 Calling the operating system . . . . .	17
4.8 Frames and scope . . . . .	18
4.9 Using C and FORTRAN routines . . . . .	20
<b>C Using R Packages</b>	<b>23</b>
C.2 Creating a package . . . . .	24
C.3 Installing R packages . . . . .	25
C.4 Converting S-PLUS libraries to R packages . . . . .	26
<b>D Script Changes when using R</b>	<b>29</b>
<b>References</b>	<b>34</b>
<b>Index</b>	<b>35</b>

# Chapter 1

## Introduction

R is a statistical system ‘not unlike S’. It is available free of charge in source-code form, and binary versions are also available for some Unix platforms and for 32-bit versions of Windows. A ‘pre-alpha’ version is available for the Macintosh. The software is distributed through the ‘CRAN’ (Comprehensive R Archive Network) set of mirror sites; to download it select a node near you from

<http://www.ci.tuwien.ac.at/R/mirrors.html>

From a user’s perspective R is very similar to S; indeed the only real documentation of the R language at present is via lists of differences from S. The R system is under rapid development and although some of these differences are deliberate, many will be removed. Note that we have said that R is similar to S, not to S-PLUS, and many of the extensions of S-PLUS do not have analogues in R at present. Nevertheless, it is possible to use R to explore most of the ideas in our book, the current major exceptions being

- Trellis<sup>TM</sup> graphics
- brush and spin plots
- library Matrix
- many non-linear models (`nlregb` and `nlme`)
- much of the optimization (`ms` and `nlminb`)
- GAMs
- factor analysis

What are (strictly) called library sections in S-PLUS are called *packages* in R. Packages for our libraries `MASS`, `nnet`, `spatial` and `class` are available through CRAN. To use Chapter 12 (survival analysis) you will need the package `survival4` (and `splines` on which it depends). To use Chapter 14 (trees) you will need package `tree`. Other packages that are needed in places are `acepack` (functions `ace` and `avas`), `akima` (function `interp`), `cluster` and `ctest` (classical statistical tests; Section 5.4). Further, some of the libraries used in the on-line statistical complements are also available for R, including

- `KernSmooth`, `boot`, `bootstrap`, `locfit`, `logspline`, `ppr`,
- `rpart`, `sm`.

We will assume that you are using a version of R not earlier than 0.64.1, and our package `MASS` is installed and invoked by

```
library(MASS)
```

at the beginning of the session. R versions of the scripts are supplied with the bundle of our libraries on CRAN, and should be used to check for any changes in commands needed whilst working through our book. (See also Appendix D.)

## Some history

R was originally written by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. Their initial experiences in implementing R are described in [Ihaka & Gentleman \(1996\)](#). Since mid-1997 there has been an ‘R core team’ of about a dozen people<sup>1</sup> who jointly develop R. R is a ‘hobby’ project in that all the core team are academic statisticians or computer scientists and all the source code is available under the GNU Public Licence. The only technical support available is via mailing lists<sup>2</sup>.

There appear to have been several motivations in developing R, some of which are conflicting. A early motivation was a belief that an S-like language could be implemented in a different, potentially better, way using ideas from the language Scheme, and those ideas form the basis of R today. Other motivations that have been mentioned were to have a statistical system for use on platforms not then supported by S-PLUS, notably Macintoshes and Linux, to have an improved version of S and to have a completely open first-class statistical language. We see tension between views that R should be completely compatible with S and that R should repair perceived short-comings in S.

## 1.1 A quick overview of R

From a user’s perspective the major difference between S and R is in the permanence of the objects created during a session. In S the objects are stored as files in the user’s file system and so are permanent. In R the objects are stored within a workspace (a region of virtual memory) and *can* be saved at the end of the session and will then be restored at the start of the next session. The user’s objects in the workspace are saved in a file named `.RData` in the current directory. If such a file is found when R is re-started, the saved objects are re-loaded into the workspace of the new session. Normally you will be asked when ending a session (with `q()`) whether you want the work saved or not. If you do not save the work, the next session starts from the last saved session (if any).

One consequence of using a workspace is that R is sometimes faster than S. Another is that if the R process crashes, the work of the session has not been saved. You can save the workspace at any time by using the command `save.image()`.

Currently R workspaces are static, that is have a fixed size, and so it easily possible to run out of workspace. The only remedies are to delete some large objects or to save the session and re-start R with a larger workspace. The workspace

---

<sup>1</sup> now including BDR

<sup>2</sup> See the CRAN web pages for how to subscribe.

size is controlled by two command-line parameters `--vsize` giving the heap size (default 6Megabytes; use `10M` to set this to 10 Megabytes) and `--nsize` giving the number of ‘cons cells’ (default 250k; each is 16 bytes in size). Usually it is the heap size that needs increasing, but loading several packages can use up all the cons cells. (If there is a problem, R will tell you which is exhausted.) You can find the current memory usage by typing `gc()`.

## 1.2 Using R under Unix

### Getting started

There is no need to prepare a directory for use with R, but it is desirable to store R sessions in separate directories.

1. Create a separate directory, say `SwR`, for this project, which we suppose is ‘Statistics with R’, and make it your working directory.

```
$ mkdir SwR
$ cd SwR
```

Copy any data files you need to use with R to this directory.

2. Start the system with

```
$ R --vsize 6M --nsize 250k
```

These options<sup>3</sup> will use about 15 Mb for the R process and will suffice to run most of the examples in our book. (A larger heap, say 30 Mb given by `--vsize 30M`, is needed for the examples of bootstrapping in the statistical complements to Chapter 5 and for Chapter 12, and will help R run faster if you have enough RAM. If you do not, it may run much slower.)

You will see a banner<sup>4</sup> similar to

```
R : Copyright 1999, The R Development Core Team
Version 0.64.2 (July 3, 1999)
```

```
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 a list.
```

```
Type    "demo()" for some demos, "help()" for on-line help, or
        "help.start()" for a HTML browser interface to help.
```

```
Type    "q()" to quit R.
```

[Previously saved workspace restored]

---

<sup>3</sup> the current defaults.

<sup>4</sup> use flag `-q` to suppress the banner.

(In this case a previous session had been saved and so is restored.)

3. At this point **S** commands may be issued. The default prompt is `>` unless the command is incomplete, when it is `+`. To use our software package issue

```
library(MASS)
```

(For users of **S-PLUS** who are used to adding `first=T`: this can be used but is not needed as packages are automatically placed first.)

4. If your version of **R** was compiled against the requisite library, command-line editing will be available, with the up and down cursor keys moving through commands (even back into earlier saved sessions), and left and right keys within the current line<sup>5</sup>.
5. It is not necessary to specify a graphics window: one will automatically be launched if graphics is needed.
6. To quit the program the command is

```
> q()
$
```

You will be asked if you wish to save the workspace image. If you accept (type `y`) and command-line editing is operational, the command history will be saved in the file `.Rhistory` and (silently) reloaded at the beginning of the next session.

Aficionados of Emacs (FSF Emacs or XEmacs) and its ESS (Emacs Speaks Statistics) package can use this to run **R**. ESS is available from

<http://franz.stat.wisc.edu/pub/ESS/>

or via any CRAN node.

## Bailing out

One of the first things we like to know with a new program is how to get out of trouble. **R** is generally very tolerant, and can be interrupted by **Ctrl-C**. (This means hold down the key marked **Control** or **Cntrl** and hit the second key.) This will interrupt the current operation, back out gracefully (so, with rare exceptions, it is as if it had not been started) and return to the prompt.

Beware that currently<sup>6</sup> interrupting a browser session *twice* in succession (by using **Ctrl-C** twice) will terminate the **R** session and not give you any chance to save the workspace image.

---

<sup>5</sup> You will probably find `man readline` tells you about the many options that are available.

<sup>6</sup> to be fixed in **R** 0.65.x.

## Getting help with functions and features

There are two ways to access the help system, closely paralleling those for S-PLUS on Unix.

(1) A help facility similar to the `man` facility. This can be invoked from the command line. For example, to get information on the function `var` the command is

```
> help(var)
```

which will put up a pager (default `more -s`; set by environmental variable `PAGER`) in the terminal window running R to view the help file. A faster alternative (to type) is

```
> ?var
```

For a feature specified by special characters and in a few other cases (one is "for"), the argument must be enclosed in double or single quotes, making it an entity known in R as a character string. For example two alternative ways of getting help on the list component extraction function, `[[`, are

```
> help("[[")
> ?"[[")
```

(2) Using R running under a Unix windowing system there is another way to interact with the help system using `help.start`.

```
> help.start()
```

to start an HTML-based help system in Netscape (which is started if not already running). If this help system is running, help requests are sent to the browser rather than to a pager in the terminal window. This help system has a Java-based search engine.

It is possible to print help pages by

```
> help(var, offline = TRUE)
```

if the system has `LATEX` and `dvips` installed and R was configured to use them. You can also use the browser to print the HTML version of the page.

## Making things easier

If there are commands that you want to have invoked for each session (like `library(MASS)`, of course), you can put these in a file called `.Rprofile`. This searched for first in the current directory then in the user's home directory, but only the first file found (if any) is read in. We used an `.Rprofile` containing

```
options(show.signif.stars = FALSE)
ps.options(horizontal = FALSE)
options(width=65, digits = 5)
.Random.seed <- c(0, 1:3)
```

when testing the R versions of the scripts for our book.

If you do not want this file to be read, start R with the flag `--no-init-file`.

## 1.3 Using R under Windows

There have been two projects porting R to Windows, but one is no longer active so we only consider that by Guido Masarotto and BDR. This runs under Windows 95, 98 and NT, and is available from CRAN in directory `bin/Windows-NT/base`. This is currently at R version 0.64.2. It provides two executables, `bin\rgui.exe` and `bin\rterm.exe`.

For normal use, launch `bin\rgui.exe` in one of the usual Windows ways. Perhaps the easiest is to create a shortcut to the executable, and set the **Start in** field to be the working directory you require, then double-click the shortcut. This will bring up its own console window from within which R can be used in almost exactly the same way as the Unix version. The other executable, `bin\rterm.exe`, can be run from an MS-DOS / Commands window but is really designed for BATCH use.

Command-line arguments such as `--vsize` and `--nsize` can be supplied as needed, most easily by typing them in the **Target** field of a shortcut after the path to the executable. It is possible to use this version of R from NTEmacs using ESS (page 4) in an almost identical fashion to under Unix; see recent versions of ESS 5.1.x for details.

The appearance of the GUI is highly customizable: see the help for `Rconsole` for details.

### Differences

There are a number of small differences from the Unix versions.

- R commands can be interrupted by `Esc` in `rgui.exe` and `Ctrl-break` in `rterm.exe`: `Ctrl-C` is used for copying in the GUI version.
- Command-line editing is always available, but is much simpler than the `readline`-based editing on Unix. For `rgui.exe`, the menu item `Help | Console` will give details.
- The commands history is saved between sessions only on `rgui.exe`.
- Using `help.start()` does not automatically send help requests to the `browser()`: use `options(htmlhelp=T)` to turn this on.
- The HTML function and package lists are not re-generated automatically by `html.start()`. The lists can be re-generated by a call to `link.html.help()` provided you have write access to the R file tree. Only packages in the standard library will be listed.
- Paths to files can be specified with either `"/"` or `"\\"`.
- The `system` command has more arguments: see the help page under `Windows` for details.

## Chapter 2

# The S Language

The R language is similar to the S language as described in [Becker \*et al.\* \(1988\)](#) and contains some of the language for modelling functions described in [Chambers & Hastie \(1992\)](#), including language extensions such as data frames. The implementation is currently incomplete and undergoing rapid development; it contains some deliberate differences. Although many of these affect programming in the language and thus features which are described in Chapter 4, the differences are all discussed in this chapter of these complements.

### 2.1 Differences between R and S

There are a number of minor differences that are not described here; some of these are in any case subject to change. One example is the function `abs` which in R is only implemented for real arguments, as trying to use it for complex ones make clear.

```
> abs(-2-0i)
Error in abs(-2 - 0+0i) : abs() unimplemented for complex; use Mod()
```

In a similar vein, there is no function `sort.list`, but `order` can be used in its place.

R regards `TRUE` and `FALSE` as the logical values but allows `T` and `F` as abbreviations<sup>1</sup> that are expanded on printing.

```
> print(T)
[1] TRUE
```

S has the converse convention.

---

<sup>1</sup> in fact they are variables with values `TRUE` and `FALSE`. `T` and `F` are not reserved words in R and so can be re-assigned to other values. For safety it is necessary to use the long forms in R code.

## Datasets

By convention datasets are not stored as R objects in the R workspace, whereas they are stored as S objects in the S library directories. (This reflects the different space priorities of using a workspace stored in a statically-sized piece of memory and storing objects in the user's file system.) Rather datasets are stored in one of a number of forms that can be loaded into R, and the function `data` is used to load datasets before first use. Be aware that large datasets will take up considerable amounts of heap memory, so the workspace will need to be sized accordingly.

## Indexing

Most of the indexing of vectors, matrices and arrays in R works exactly as in S. However, there is an exception. In R

```
> x <- 1:10
> x[12]
[1] NA
> x[-12]
Error: subscript out of bounds
```

whereas `x[-12]` gives `1:10` in S.

## Lists

This subsection describes the position from R version 0.63; this differs considerably from earlier releases of R.

The semantics of assigning `NULL` are different. For example,

```
Empl <- list(employee="Anna", spouse="Fred", children=3,
             child.ages=c(4,7,9))
Empl["spouse"] <- NULL
```

removes the `spouse` component in R but does nothing in S, whereas

```
Empl["spouse"] <- list(NULL)
```

sets the `spouse` component to `NULL` in both R and S, and

```
Empl[["spouse"]] <- NULL
```

removes it in both.

## Matrices and data frames

R allows matrices and arrays with some elements of their `dim` attribute as zero; S does not. For example, if `x` is a matrix, `x[, F]` is a matrix with 0 columns. This is another example where the correct test for existence is `length(x) > 0` not `!is.null(x)`.

R has a rather stricter interpretation of a matrix than S; `is.matrix` is true for a data frame in S but false in R.

## Random-number generator

The default random-number generator in R is completely different from that in S. It is not of the same quality, being the combination of three short-period congruential generators suggested by [Wichmann & Hill \(1982\)](#). This needs a seed for each generator, and so the `.Random.seed` in R is a vector of length 3. Its elements should be strictly positive and less than 30269, 30307 and 30323 respectively. From R 0.63.1 better-quality alternatives are available and can be set by a call to `RNGkind`; see the help page for `RNGkind` for details of what is currently available. (These include a version of the random number generator used in S, "Super-Duper", and "Marsaglia-Multicarry", both of which we recommend over Wichmann-Hill for serious simulation work. Note that "Super-Duper" is *not* precisely the same as the generator used in S-PLUS.)

Unlike S, there is no initial `.Random.seed`; if when the random-number generator is used `.Random.seed` is not found, a value is created from the system clock.

There is no function `set.seed`.

## Formulae

Whereas in S you may use `lm(y ~ x^3)`, in R you have to use `lm(y ~ I(x^3))`. In R `y ~ x + 0` is an alternative to `y ~ x - 1` for specifying a model with no intercept. Models with no parameters at all can be specified by `y ~ 0`.

The default contrasts are different in R,

```
options(contrasts=c("contr.treatment", "contr.poly"))
```

This is in general a good idea (as its frequent appearance in our book shows), except for aov models.

Formulae for nested models (such as those on page 195) are represented differently: `%in%` is not used in R.

The handling of `-` mentioned on page 203 is different in R: `a*b - b` is equal to `a + a:b`.

## Frames and scope

This is the area of the most fundamental differences between the languages. See Section 4.7 for some details.

## Sugar

There are several extra utility functions that can be used to make code more readable. Some examples are

```
NROW, NCOL      like nrow and ncol but will also work for vector objects.
```

<code>rownames</code>	give or set the first and second component of the <code>dimnames</code> of a matrix, array or data frame. (A missed opportunity here: these do not work for vectors.)
<code>rownames&lt;-</code>	
<code>colnames</code>	
<code>colnames&lt;-</code>	
<code>apropos</code>	a <code>find</code> -like function using regular expressions.
<code>case.names</code>	extract the names of cases (observations) from a fit.
<code>chol2inv</code>	inverse from Choleski decomposition.
<code>choose</code>	binomial coefficients (as in <code>S-PLUS 4.x</code> and <code>5.x</code> ).
<code>digamma</code>	digamma and trigamma functions (and also other derivatives) for real arguments. (The functions in package <code>MASS</code> also accept complex arguments.)
<code>trigamma</code>	
<code>drop.terms</code>	drop terms in a <code>terms</code> object.
<code>gl</code>	a 'generate levels' function based on <code>%GL</code> in <code>GLIM</code> .
<code>gsub , sub</code>	substitute regular expressions in strings.
<code>IQR</code>	Inter-Quartile Range, as in library <code>MASS</code> for <code>S-PLUS</code> .
<code>is.R</code>	R function which checks if the code is being run in R. See page 27 of these complements or its help page for how to use it.
<code>mat.or.vec</code>	create a matrix or vector.
<code>nlevels</code>	the number of levels of a factor.
<code>variable.names</code>	extract the names of variables from a fit.

## 2.11 Customizing your R environment

There are many fewer options that can be set by `options` than in the table on page 65. Of those, `width`, `digits`, `echo`, `prompt` and `continue` are available, as well as the `S` options `contrasts` and `na.omit` that refer to model matrices. However, `echo` seems to have essentially no effect in R; commands are echoed if input is from a file unless `--quiet` was used, and are never echoed if input is from a terminal (or terminal-like connection).

There are some further options specific to R:

<code>show.signif.stars</code>	A logical: should significance stars be shown in tables of <i>t</i> -ratios and anova tables?
<code>printcmd</code>	The command to be used for printing, e.g. <code>"lpr"</code> .
<code>papersize</code>	The papersize, defaulting to the ISO standard <code>"a4"</code> .
<code>device</code>	The default graphics device.
<code>browser</code>	The default HTML browser for <code>help.start</code> (Unix).

Some of these can also be set in environmental variables, for example `R_PRINTCMD` and `R_PAPERSIZE`. The variable `RLIBS`<sup>2</sup> controls the default search path for packages (see page 24 of these complements).

On the Unix version, environment variables to be used with R can be set in the file `~/Renvirom` as `sh` commands, for example (R 0.64.1 or later)

<sup>2</sup> `RLIBS` from R 0.65.x.

```
R_VSIZE=10M
R_NSIZ=400k
RLIBS=/ext/R/library
R_LIBS=/ext/R/library
export R_VSIZE R_NSIZ RLIBS R_LIBS
```

This also works on the Windows version of R 0.64.2 or later, although the last line is not needed.

We have already mentioned the use of `.Rprofile` on page 5. It is also possible to have a system-wide profile file which is read before the user's file (if any). The system profile should be stored in `${RHOME}/etc/Rprofile`; you can suppress reading this by the flag `--no-site-file`.

Functions `.First` and `.Last` can be used just as in S-PLUS, but normally a `.Rprofile` file will be more convenient than `.First`, especially as these functions will only be found if a workspace is restored or if they are defined in a `.Rprofile` file.

## 2.13 BATCH operation

Unix versions of R have a BATCH command: use

```
R BATCH [options] infile [outfile]
```

If `outfile` is missing it defaults to `infile.Rout`, stripping any `.R` extension. Unlike Splus BATCH this does not run the job in the background; use shell facilities (usually ending the line by `&`) to do this.

Whenever R detects that input is coming from a file, the input is automatically echoed to the output.

The Windows version has a front-end `rterm.exe` that can be run from a shell in a terminal window, and can be called from a batch (`.bat`) file or shell script to do equivalent things to R BATCH.

## Chapter 3

# Graphical Output

### 3.3 Enhancing plots

#### *Mathematics in labels*

Labels on axes and plots in most of the high-level graphics can be expressions rather than character strings. If they are expressions, they are evaluated as a mathematical formula and superscripts, subscripts and greek letters will be interpreted. More precisely

- unary and binary operators are interpreted as one would expect, except that  $x * y$  is reproduced as  $xy$ . Use `==` for an equals sign, `%~%` for  $\approx$ , `%==%` for  $\equiv$  and `%prop%` for  $\propto$ .
- `x[2]` is a subscript and `x^2` is a superscript. Use `{...}` for invisible groupings such as  $e^{\sin(x)}$
- lower and upper-case greek letters (such as `pi` or `Omega`, for example) will be interpreted, as will `degree`, `minute`, `second` (as in  $2^\circ 10' 33''$ ) and `infinity`.
- there are accents `hat`, `tilde`, `bar`, `widehat` and `widetilde`. The last three stretch to cover the expression.
- the following functions will be interpreted

```
hat, bar, sqrt, abs, frac, sum,
product, integral, union, intersection
```

In particular, `sqrt(x, n)` indicates an  $n$ th root.

- the function `list` will produce a list separated by commas.
- `...` will be interpreted as three dots: use `cdots` or `ldots` to force centring or base-line alignment.
- the functions `frac` (equivalently `over`) and `atop` produce two-line displays from two expressions, separated by a line for `frac`: these can be used recursively.

- functions `group` and `bgroup` will produce groupings, the latter with variable-sized delimiters, as in

```
bgroup("(", atop(n, k), ")")
```

- the function `paste` will juxtapose expressions: use `~` to separate expressions (and this can be repeated to supply more space).
- the functions `plain`, `bold`, `italic`, `bolditalic` change the fonts of their arguments, and `displaystyle`, `textstyle`, `scriptstyle` and `scriptscriptstyle` change their sizes.

It is normally necessary to avoid the expression being evaluated by enclosing it in a call to `expression`. (This does not always work, but most of the errors in evaluating expression labels have now been found.)

Full details can be found in [Murrell & Ihaka \(1999\)](#).

### 3.7 The R colour model

In S-PLUS, colours for plots are assigned by number, and the mapping of numbers to colours is device-specific (and can be altered after the plot is drawn on most graphics devices). In R, the colours are specified by name, and numbers are mapped to named colours by a *palette* defined by R (rather than by the graphical device).

The list of known colours<sup>1</sup> is returned as a character vector by `colors()`, and the current mapping from numbers to colour names is given by a call to `palette()`. The default is

```
> palette()
[1] "black"  "red"    "green3" "blue"   "cyan"   "magenta"
[7] "yellow" "white"
```

Colour numbers are reduced modulo the length of the palette; for example, with the default palette colour 11 is `green3`.

Palettes can be set by a call to `palette`: see `?palettes` for ways to create palettes of contiguous ranges of colours. New colours can be created as hexadecimal red-green-blue triples with names starting with `#`, for example `"#BFBFBF"`; functions `rgb`, `hsv` and `gray` help to create such colours.

Note that the same colour space is used for all purposes, unlike S-PLUS which has separate spaces for lines, text, polygon fills and images.

The function `par` will usually return a name for its parameters such as `col`; if a number is required (for example to cycle through the colours) use

```
col <- par("col")
if (!is.numeric(col)) col <- match(col, palette())
```

---

<sup>1</sup> which will be familiar to users of X11 displays.

A graphical device has several colour settings. As well as `col` (the current plotting colour for lines, text and fills on the plot) there are `bg`, the background colour to which figure regions will be cleared and `col.axis`, `col.lab`, `col.main` and `col.sub` which are the colours used for axes, axis labels, plot main titles and plot sub-titles respectively. (There are also `cex` and `font` settings for these categories.)

## Chapter 4

# Programming in S

### 4.5 Editing, correcting and documenting functions

#### Editing R functions and objects

All the methods described for S-PLUS work under R, except `fix()` with no argument.

#### Locating and correcting errors

R has much less extensive facilities for debugging. The two main tools are `traceback` and `browser`: there is no equivalent of `debugger` or `inspect`. Here is a simple use of `traceback`.

```
> data(iris)
> library(MASS)
> lda(species ~ ., iris)
Error: Object "species" not found
> traceback()
[1] "model.frame.default(formula = species ~ ., data = iris)"
[2] "eval(m, sys.frame(sys.parent()))"
[3] "lda.formula(species ~ ., iris)"
```

The problem is that the variable is called `Species` in data frame `iris`.

Let us consider the example of the *t*-test function of page 126 studied on pages 141ff.

```
ttest <- function(y1, y2, test = "two-sided", alpha = 0.05)
{
  n1 <- length(y1)
  n2 <- length(y2)
  ndf <- n1 + n2 - 2
  s2 <- ((n1 - 1) * var(y1) + (n2 - 1) * var(y2))/ndf
  tstat <- (mean(y1) - mean(y2))/sqrt(s2 * (1/n1 + 1/n2))
  tail.area <- switch(test,
    "two-sided" = 2 * (1 - pt(abs(tstat), ndf)),
    lower = pt(tstat, ndf),
```

```

    upper = 1 - pt(tstat, ndf),
    {
      warning("test must be 'two-sided', 'lower' or 'upper'")
      NULL
    }
  )
  list(tstat = tstat, df = ndf,
       reject = if(!is.null(tail.area)) tail.area < alpha,
       tail.area = tail.area)
}
x1 <- round(rnorm(10), 1); x1
x2 <- round(rnorm(10) + 1,1); x2
x12 <- c(x1, x2)
sam <- factor(rep(0:1, c(length(x1), length(x2))))
ttest(x12[sam == 1], x12[sam == 2])

```

In R this does not give an error, just NA as the answer. So we have no traceback to help us know where to begin. This is a case for `browser`. Edit the function (using `fix(ttest)`) and insert a call to `browser()` at its head. Use `n` at the browser prompt to step through the code. Press return to advance one step (`c` resumes running).

```

> ttest(x12[sam == 1], x12[sam == 2])
Called from: ttest(x12[sam == 1], x12[sam == 2])
Browse[1]> n
debug: n1 <- length(y1)
Browse[1]>
debug: n2 <- length(y2)
Browse[1]>
debug: ndf <- n1 + n2 - 2
Browse[1]>
debug: s2 <- ((n1 - 1) * var(y1) + (n2 - 1) * var(y2))/ndf
Browse[1]>
debug: tstat <- (mean(y1) - mean(y2))/sqrt(s2 * (1/n1 + 1/n2))
Browse[1]> s2
[1] NA
Browse[1]> var(y1)
[1] 0.2773333
Browse[1]> var(y2)
[1] NA
Browse[1]> y2
numeric(0)
Browse[1]> c
$tstat
[1] NA
$df
[1] 8
$reject
[1] NA
$tail.area
[1] NA

```

This can often be an effective way to debug, although sometimes one has to do two passes, one to locate the error and another to examine the variables used.

Notes: If you wish to examine variables `n` or `c` you need to explicitly print them. The list of local variables can be found by `ls()`. Beware of using `ctrl-C` in the browser: you can end up terminating the R process if it is used more than once.

There are functions `debug` and its companion `undebug` that effectively add and remove a call to `browser` at the top of the function. These are useful for quick debugging, but we find that adding a call to `browser` at the most appropriate place in a long function can be more productive.

### Creating a help document

A help document is created with `prompt` just as for S-PLUS. However, the language used is rather different, and is about the only thing documented in the R manual. As with the `nroff` dialect used for S on Unix, one can learn a great deal by copying examples. The sources for the system help files are in

```
.../src/library/base/man
```

Unlike S, only pre-processed help pages are stored in the installed system.

The Emacs ESS package mentioned on page 4 has a mode for editing R help files, which conventionally have suffix `.Rd`.

## 4.6 Calling the operating system

The main function for accessing the operating system is `system` rather than `unix`, `win3` or `dos`. Its syntax under Unix is

```
system(command, intern = FALSE)
```

where the argument `command` is a character string. If `intern` is false, the `system` call is used to invoke the command which may produce output, and the return value is the exit status returned by the operating system. If `intern` is true, the output is collected line-by-line into a character vector which is the return value.

The `date` function provides an example of using `system` (under Unix):

```
> date
function ()
{
  system("date", intern = TRUE)
}
> date()
[1] "Saturday July 11 21:31:36 BST 1998"
```

## 4.8 Frames and scope

This is the area of the most fundamental differences between the languages. What are called *frames* in S are called *environments* in R, but the parallel is only a rough one. Functions such as `sys.parent` have a different meaning; they return the *number* of the environment, which is converted by `sys.frame` into an environment. Thus the typical S construction

```
eval(expression, sys.parent())
```

is rendered in R as

```
eval(expression, sys.frame(sys.parent()))
```

(From R 0.65.0 this can be written as `eval(expression, parent.frame())`.)

Using `eval` can be tricky, especially in converting code designed for S. Note that the first argument is evaluated in the calling environment, then its *value* is evaluated in the specified environment. The evaluated argument should be an expression or call, so one needs, for example

```
eval(expression(x^2), sys.frame(sys.parent()))
```

to ensure that the first argument has not already been evaluated or attempted to be evaluated. Alternatively, `evalq` can be used, which quotes its first argument. In R, `quote` is often used to pass expressions down unevaluated.<sup>1</sup> As of version 0.63, `eval` and `evalq` have arguments `expr`, `envir`, `enclos`, the second and third being environments analogous to the `local` and `parent` frame arguments of S's `eval`.

The equivalent of the working database is the *global environment* `.GlobalEnv`, and in general environments are given (hexadecimal) addresses or names not numbers. Functions such as `assign`, `get` and `exists` have an argument `envir` which replaces `where` and `frame`.

Perhaps even more important is the difference in the scope rules. R is said to be *lexically scoped* and this provides ways to avoid the need of an analogue of placing objects in frame 1 to make them visible. We refer the reader to [Ihaka & Gentleman \(1996\)](#) for a detailed discussion of the differences, merely showing two examples.

The following code<sup>2</sup> prints 123 in S and 100 in R.

```
y <- 123
f <- function(x)
{
  y <- x*x
  g <- function() print(y)
  g()
}
f(10)
```

---

<sup>1</sup> It is not a synonym for `substitute` as claimed on the help page.

<sup>2</sup> from [Ihaka & Gentleman \(1996\)](#).

Here the R interpretation seems more natural, and this type of construction has caught out many S programmers (although, fortunately, the result is more often that `y` is not found or is of the wrong length). Ihaka & Gentleman conclude that

‘The real difference is that there is this associated environment that can be used to store information between function calls.’

However, care is needed, and the next example<sup>3</sup> comes as a complete surprise to S programmers who find the searching of parent frames attractive. For

```
y <- 456
g <- function() print(y)
y <- 123
f <- function(x) {y <- x*x; g()}
f(10)
```

returns 123 in R! The environment of a function is established when the function is *defined*, not when it is *invoked*. Why then does the second example not give 456? Because `y <- 123` has been evaluated in that environment, and changed the association between symbols and values. To see what is in the environment, use the function `environment`. Thus

```
> g <- function() print(y)
> environment(g)
<environment: R_GlobalEnv>
> ls(envir=environment(g))
[1] "f" "g" "y"
```

so `g` has the global environment rather than a specific one. However

```
f <- function(x)
{
  y <- x*x
  g <- function() print(y)
  print(ls(envir=environment(g)))
  g()
}
> f(10)
[1] "g" "x" "y"
[1] 100
```

has the ‘frame’ in which the function `g` was defined as its environment.

Lexical scoping can be used to good effect, illustrated by the examples in the R FAQ<sup>4</sup>. Our experience is that it is hard to be expert in the use of both R’s and S’s scoping rules.

Like S, R has a ‘super assignment’ operator `<<-`, but with different semantics. In S `<<-` replaces (or creates) a value in the working directory. In R `y <<- value` replaces the value of `y` in the environment (excluding the local

<sup>3</sup> not from Ihaka & Gentleman!

<sup>4</sup> available online at CRAN at <http://www.ci.tuwien.ac.at/~hornik/R/>.

one) from which its value would be taken, or the global environment if no value would have been found. Note that it will be quite rare for the assignment to occur other than in the global environment, as this can only occur if `<<-` is used in a function nested within a function or that was returned as the value of another function. (See [Ihaka & Gentleman \(1996\)](#) for an example.)

R follows the *lazy evaluation* principles of S, but with a few subtle differences. Thus in `hist.default` R has to force evaluation of the labels:

```
hist.default <-
function (x, breaks, freq = NULL, probability = !freq,
        include.lowest = TRUE, right = TRUE, col = NULL,
        border = par("fg"),
        main = paste("Histogram of", deparse(substitute(x))),
        xlim = range(breaks), ylim = range(y, 0),
        xlab = deparse(substitute(x)), ylab, axes = TRUE,
        plot = TRUE, labels = FALSE, ...)
{
  if (!is.numeric(x)) stop("hist: x must be numeric")
  main
  xlab
  n <- length(x <- x[!is.na(x)])
  ....
}
```

as `substitute` does not evaluate its argument, and `x` is changed in the body of the function before `main` and `xlab` are used. S keeps track of the frame in which lazy evaluation is to be performed (here for an argument in the parent frame) in a different way from R.

## 4.9 Using C and FORTRAN routines

The storage mode `single` does not exist in R, and correspondingly float (C) and real (FORTRAN) types cannot be used in interfacing code to R.

The only form of dynamic loading in R is loading of shared libraries (Unix) or DLLs (Windows). A shared library or DLL can be loaded by a call to the function `dyn.load`<sup>5</sup>, or, for a package, `library.dynam`.

Under Unix the compilation process is done in two phases. First R `COMPILE` is used to compile the C and/or FORTRAN files; then R `SHLIB` is used to create the shared library. However, `SHLIB` will compile as required. Thus for the `horner.c` example in Section 4.9 of the on-line programming complements we can use

```
R SHLIB horner.c
```

to build a shared library<sup>6</sup> `horner.so` and then dynamically load this by the command `dyn.load("horner.so")`. We can test this by

<sup>5</sup> which is the analogue of `dyn.load.shared` in S-PLUS 3.x or `dyn.open` in S-PLUS 5.x rather than of `dyn.load`.

<sup>6</sup> the extension may differ on your version of Unix and will be `.dll` on Windows.

```

polynom <- function(x, b)
{
  m <- as.integer(length(x))
  n <- as.integer(length(b)-1)
  storage.mode(x) <- "double"
  p <- x
  storage.mode(b) <- "double"
  .C("poly", m, val = p, x, n, b)$val
}
dyn.load("horner.so")
> mat <- matrix(1:9,3,3)
> polynom(mat, -1:1)

```

There is a fair degree of compatibility between the interfaces that facilitates porting code written for S. The compatibility header file `S.h` defines `S_alloc`, `S_realloc`, `Calloc`, `Free`, `PROBLEM`, `RECOVER`, `WARNING`, `unif_rand`, `norm_rand`, `seed_in` and `seed_out`, as well as `F77_NAME` and `F77_CALL`. Using `NAOK=T` in a `.C` or `.Fortran` call allows in NaNs and infinite values in the way `special sok=T` does in S-PLUS.

The macros to handle missing values are different, and in R NaN is not regarded as NA by the macros. However, the macro `ISNAN` will be true for either NaN or NA, so an R version of the `ourdist` C code might be

```

/* R version */
#include <Arith.h>
#include <math.h>

void ourdist(double *x, long *nin, long *pin, double *res)
{
  int i, j, k, den, n = *nin, p = *pin;
  double item, tmp;

  for (i = 0; i < n-1; i++)
    for (j = i+1; j < n; j++) {
      den = 0; tmp = 0.0;
      for (k = 0; k < p; k++)
        if (!ISNAN(x[i + n * k]) && !ISNAN(x[j + n * k]))
          {
            den++;
            item = x[i + n * k] - x[j + n * k];
            tmp += item * item;
          }
      if (!den) tmp = NA_REAL;
      else tmp = sqrt(tmp * p / den);
      *res++ = tmp;
    }
}

```

In principle, FORTRAN input/output can be used directly from subroutines compiled and dynamically loaded into R; note that it may be necessary to link

against the Fortran libraries explicitly to get this to work. The functions such as `INTPR` and `DBLEPR` are provided for compatibility, and can be very useful.

The analogue of `call_S` is `call_R`; `call_S` exists for compatibility.

There is no analogue of `XERROR`.

### Under Windows

There are no pre-packaged aids to building a suitable DLL under Windows, except as part of installing a package (see page 25). Any compiler can be used to build a DLL following the `cdecl` conventions, and we have tested most of the examples given in our Programming Complements. The same restrictions apply as for S-PLUS; care is needed when calling functions and global variables in the R engine. Many R packages do do this, and it is then easiest to use the preferred compilers, a `mingw32` version of `gcc/g77`. Details are given on page 25.

Our experiments suggest that the preferred compilers use the same calling conventions as Visual C++, so it seems relatively easy to use that when calling entry points in the R engine. For example, we can call the random number generator by using file `VCrnd.c`:

```
__declspec(dllimport) double unif_rand();
__declspec(dllimport) void seed_in();
__declspec(dllimport) void seed_out();

__declspec(dllexport) void urand(long int *m, double *p)
{
    long i;
    seed_in((long*)0L);
    for (i = 0; i < *m; i++) p[i] = unif_rand();
    seed_out((long*)0L);
}
```

We can compile and test this by Visual C++ 4.2 using

```
lib /def:R.def
cl /MT /Ox /D "WIN32" /c VCrnd.c
link /dll /out:VCrnd2.dll VCrnd.obj R.lib

> .Random.seed <- c(1, 1:3)
> .C("urand", as.integer(4), x=double(4))$x
[1] 0.5641106 0.2932388 0.6696743 0.9174765
```

which is the same answer as `runif(4)` starting from that seed. (The file `R.def` is in the source and package-building distributions of R for Windows; the first line creates an import library.)

## Appendix C

# Using R Packages

In S-PLUS the official terminology is that MASS is a library *section* contained in a directory, the *library*. R calls library sections *packages*, but they are used in exactly the same way. To find out what packages are available on your system use

```
> library()
Packages in library '/ext/R/current/lib/R/library':

KernSmooth      functions for kernel smoothing corresponding to book:
                  Wand, M.P. and Jones, M.C. (1995) "Kernel Smoothing"
MASS             Main Library of Venables and Ripley's MASS
acepack         ace() and avas() for selecting regression
                  transformations
akima           interpolation of irregularly spaced data
base            The R base package
boot            Bootstrap S-Plus Functions
bootstrap       Functions for the Book
                  "An Introduction to the Bootstrap"
class           Functions for classification
cluster        Functions for clustering
ctest          Classical Tests
eda            Exploratory Data Analysis
integrate       numerical integration
lme            Linear mixed effects library
locfit         Local Regression, Likelihood and Density Estimation.
logspline      logspline density estimation
lqs            Resistant Regression and Covariance Estimation
modreg         Modern regression: smoothing and local methods
mva            Classical Multivariate Analysis
nnet           Software for feed-forward neural networks with a
                  single hidden layer and for multinomial
                  log-linear models.
ratetables     US mortality data tables
rpart          Recursive partitioning.
spatial        functions for kriging and point pattern analysis
splines        Regression Spline Functions and Classes
stepfun        Step Functions, including Empirical Distributions
survival4      Survival analysis [needs library(splines)]
tree           Classification and regression trees.
```

Some of these are standard (`base`, `eda`, `lqs`, `modreg`, `mva`, `stepfun`) and the others have been installed as extensions to R. To find out what is in a library use the `help` argument,

```
> library(help=eda)
line           Robust Line Fitting
medpolish      Median polish
smooth        Median smoothing
```

and to attach the library use `library(name)`.

Packages in places other than the standard library may be used by specifying the argument `lib.loc` to `library`<sup>1</sup> or by setting the variable `.lib.loc`. This should be a character vector giving the locations of *all* the libraries to be searched, including the system library (whose location is in `.Library`). The variable `.lib.loc` is initialized to the (colon-separated) values given in the environment variable `RLIBS` and `.Library`, and setting `RLIBS` is the preferred way to change the library search path. You will not find the help for functions in packages in private libraries unless `RLIBS` (or `.lib.loc`) is set.

Unlike S-PLUS, attaching a package loads code and so can use up precious workspace resources. You may want to defer loading packages until they are definitely needed, and to remove them afterwards by, for example,

```
detach(package:survival4)
```

## C.2 Creating a package

A package in R contains files `DESCRIPTION`, `TITLE`, `INDEX` and directories `R`, `data`, `src` and `man`. The R source files go in `R` and have one of the extensions `.R`, `.S`, `.q`, `.r` or `.s`. C and FORTRAN source files go in `src`, together with a `Makefile` if required. The `man` directory should contain R documentation files with a `.Rd` extension. Datasets are stored in `data` directory as R code (`.R`), matrices to be read by `read.table(file, header=T)` (`.tab`, `.txt` or `.csv`) or saved R code (`.rda`).

When a package is attached, the commands in the function `.First.lib` is executed if this exists in the package. This is typically used to load compiled code, by something like

```
.First.lib <- function (lib, pkg)
  library.dynam("MASS", pkg, lib)
```

Note the different orders of the arguments!

For further details see `?library`. Do remember to write the documentation files such as `DESCRIPTION` (see page 26), `TITLE`, and `data/00Index!`

Wherever possible the R code in help pages should be directly executable.  
Running

```
R CMD check pkgname
```

after installation (Unix) will provide a check of this.

<sup>1</sup> Note that unlike S-PLUS, the system library is not searched when `lib.loc` is specified

## C.3 Installing R packages

This is straightforward under Unix. A package is usually distributed as a file with extension `.tar.gz` or `.tgz`. First unpack this in a temporary file area by

```
gzcat name | tar xvf -
```

or, if you have the GNU version of `tar`,

```
tar zxvf name
```

This should create a single directory, say `libname`. Then install the package by

```
R INSTALL libname
```

That form uses the system library: if you need or wish to use a private library use

```
R INSTALL -l path/to/library libname
```

Installation is even easier under Windows; change directory to the library directory under `RHOME` and unzip the pre-packaged zip file. Some Linux systems also have pre-packaged compiled libraries.

### Building from a source-code library under Windows

First collect the tools that you need.

EITHER

The `cygwin-B20` set of tools available from <http://sourceware.cygnum.com/cygwin> and several mirrors. If you need to compile FORTRAN you will need `egcs-1.1-f77.tar.gz` from that page (or directly from <http://www.xraylith.wisc.edu/~khan/software/gnu-win32/>) plus the Fortran library `libg2c.mingw32.a.bz2` from `CRAN/bin/ms-windows/Windows-NT/etc`. It is preferable to replace the `cygwin` compilers by the port of `gcc-2.95` available at Mumit Kahn's site.

OR

The `mingw32` port of `gcc-2.95` from <ftp://ftp.xraylith.wisc.edu/pub/khan/gnu-win32/mingw32/>, the `ld.exe` and `cygwin1.dll` from the B20 set<sup>2</sup> in your path ahead of the `mingw32` binaries. (The ports of `egcs-1.1` or `egcs-1.1.2` can also be used.)

Suitable versions of `rm`, `sed`, `mkdir`, `echo`, `cp` and `cat`, for example from the `cygwin` set, a copy of `perl5`, available via <http://www.perl.com/CPAN/ports> and `groff`, available at <http://www.itribe.net/virtunix/groff-1.10nt.zip>.

---

<sup>2</sup> available separately in `CRAN/bin/ms-windows/Windows-NT/etc`

All of these need to be installed and in your path, and the appropriate environment variables set. Then `cd RHOME/src/gnuwin32` and run `make libR.a` which will take several minutes<sup>3</sup>.

For each package you want to install, unpack it to a directory, say `mypkg`, in `RHOME/src/library`, `cd RHOME/src/gnuwin32` and make `pkg-mypkg`. To test the package, run `make pkgcheck-mypkg`.

## C.4 Converting S-PLUS libraries to R packages

If a package does not use compiled code, the conversion is in principle straightforward. Suppose we wish to convert library `mytest` which has files

```
$ ls
mytest.q  README    test1.d  test2.d
```

Create directories `R` and `man` and move the S code files (here `mytest.q`) to directory `R` and the help files `*.d` to `man`. Then use (in `cs`, something similar in other shells)

```
cd man
foreach f (*.d)
  R CMD Sd2Rd $f > {$f:r}.Rd
end
rm *.d
cd ..
R CMD Rdindex man/*.Rd > INDEX
```

Then look at `README` and edit `INDEX` to include any information you want `library(help=mytest)` to give. (This lists `INDEX` in R, `README` in S-PLUS.) Finally create a suitable one-line file `TITLE` for use by `library()` and a `DESCRIPTION` file along the lines of

```
Package: mytest
Version: 0.2-1
Author: An author <author@dept.domain.dom>.
Description: Something about the contents
License: Unlimited non-commercial use.
```

Include a `Depends:` line after description if this package depends on other packages or on a particular version of R.

The package can then be installed (under Unix) by

```
R INSTALL .
```

Now test the code. Changes that need to be made are conventionally noted in a file `PORTING`. Things to watch for include

<sup>3</sup> 13 minutes with `cygwin` and 5 with `mingw32` on BDR's 133MHz Pentium 32Mb laptop. With more memory and a fast disc system the times can be less than 1 minute, and on a networked file system this can take many minutes.

- Missing functions (e.g. `sort.list`, `rep.int`).
- Functions that have been ‘improved’ in R in incompatible ways, such as
  - Different argument names (for example `ace` has `lin` not `linear`) and different ordering of the arguments (`hist`).
  - Arguments with a different meaning, such as `start` in `glm` and `border` in `polygon`.
  - Different return components: for example `family` returns `link` in S but `linkfun` in R.
  - Return components with the same name but a different structure or meaning, such as `assign` in `lm` objects.
- Incomplete functionality (e.g. `locator` lacks the `type` argument, and `uniroot` has fewer arguments under R).
- `terms` objects are different: the main part is a formula rather than an expression, and the `"variables"` attribute is a list not a vector. Further, the function `terms` is generic and is by no means compatible with that in S. In particular, calling `terms` on a `terms` object does something useful in S but not in R.
- Assuming that `.Random.seed` will exist.
- Tests involving single-precision quantities such as `Machine$single.eps`: this does not exist in R.
- Expecting `par("col")` to be numeric (see Section 3.7).
- Check any use of `eval`: you may need `sys.frame(sys.parent())` rather than `sys.parent()`.
- Plot labels and titles may need to be evaluated if they are produced by `deparse(substitute(x))` and `x` is changed.
- Assignments to frame 1 need to be studied: they may not be necessary or they may need to be written in a different way or as an assignment to `.GlobalEnv`.
- Any attempt to manipulate functions as R objects will have to be done differently, using functions `body` and `formals`.
- The assignment `fn(a) <- b` is computed as `(fn<-)(a, b)` in S but `(fn<-)(a, value=b)` in R, so the last argument of all assignment functions in R must be named `value`.

If you want to maintain versions of code that works with both R and S, the function

```
is.R <- function ()
  exists("version") &&
  !is.null(v1 <- version$language) && v1 == "R"
```

is in R and can be used in S-PLUS.

## Data

Many libraries include datasets to run their examples. If you have those, create a directory `data` and put the datasets in that directory in a form that the R function `data` can read them. Tables to be read by `read.table(file, header=T)` should have extension `.tab`. For other datasets it is probably easiest to read them into R in whatever way would be used by S, and then dump them by one of

```
dump("name", "name.R")
save(name, file="name.rda", ascii=T)
```

Do try to be consistent and match the names, or `data(name1)` could load a dataset `name2`!

## Compiled code

The first thing to note is that R does not have storage mode `single`, so any `single` variables in the interface to S will need to be converted to `double`. If you can, it is often helpful to test the converted version under S-PLUS.

Create a directory `src` and move the C and/or FORTRAN source files to `src`. You will need to ensure that the code is loaded, so add to the `src` directory a file, conventionally `zzz.R`, containing

```
.First.lib <- function(lib, pkg)
  library.dynam("mytest", pkg, lib)
```

You can then try installing, which will attempt to compile the source code and create `mytest.so`. If this fails, it may be easiest to debug this (under Unix) by

```
cd src
R SHLIB -o mytest.so *.{cf}
```

It is not often necessary to write a `Makefile`, but one can be included in the `src` directory.

A comment for users of LINPACK in their packages. The R binary links in some LINPACK routines, although fewer than S. The full BLAS library can be assumed to be available for a library, as it will be included if necessary at link time.

There is a trap for package porters prior to R 0.65. Under Unix, the `dyn.load` does not check immediately that all the symbols are satisfied, but crashes R (with an error message) when the code is called. So check carefully that all the symbols will be satisfied (for example by using 0.65.0 or later to test your code).

If you want to maintain versions of code that works with both R and S-PLUS, the macro `USING_R` is defined in R (but not S-PLUS).

## Appendix D

# Script Changes when using R

Only relatively small changes are needed in most chapters to run our examples under R. One common theme is that the datasets are not initially loaded into R, and have to be made available by a `data` command, for example `data(hills)`. Beware that some datasets have different names: `iris` is `iris3` and `swiss.x` and `swiss.fertility` have been combined into dataset `swiss`<sup>1</sup>.

Some of the functions are in the supplied packages `modreg` and `mva`; in particular `loess` is in package `modreg`.

Another theme is the absence of Trellis<sup>TM</sup> graphics.

For precise details see the scripts supplied with the R versions of our libraries on CRAN.

### Chapter 1

On page 11, `trellis.device()` should be omitted.

On page 12, `contourplot`, `wireframe` and `levelplot` are Trellis functions, and `hist2d` is also missing.

On page 13, replace `splom(~ hills)` by `pairs(hills)`. Functions `ltsreg` (pages 13 and 16) and `rreg` (page 16) do not exist in base R but an emulation of `ltsreg` is provided by package `lqs`.

Function `usa` (page 17) is missing.

### Chapter 2

R has no `ordered` replacement function (page 26).

Function `sort.list` does not exist, but may be replaced by `order`.

There is no library/package `Matrix`.

---

<sup>1</sup> with numbers entered to a higher precision.

### Chapter 3

As there was no function `ts.union` prior to R 0.65.0, dataset `lung.deaths` (page 74) is supplied in package `MASS`.

There is no `par` value "cxy", but we can use (page 74)

```
text(loc, total + yinch(par("csi")), total, cex=0.7)
```

The `subplot` function does not exist, and `split.screen` behaves somewhat differently (in particular, in clearing screens).

The dataset `swiss` is handled differently in R and can be used in place of `swiss.df` on page 87.

None of Sections 3.5 and 3.6 is relevant.

### Chapter 4

Dataset `iris` is called `iris3` (pp. 118–9).

Functions `crosstabs` (page 120), `by` and `merge` (pp. 124–5) do not exist in R.

Function `integrate` (pp. 150–3) is in package `integrate` (with return value `finest` not `integral` and automatic vectorization).

### Chapter 5

Function `qqmath` (page 165) is a Trellis function.

Argument `scale` to `stem` acts differently: we could use `stem(abbey, scale=0.4)` (page 172). The `stem` plots come out rather differently as the outliers are not excluded in R.

Function `bwplot` (page 172) is a Trellis function.

The functions of Section 5.4 are in package `cctest`. However, there is no `cdf.compare` (page 177).

The syntax of `hist` is slightly different, and on page 178 we need

```
rep(hist(eruptions, breaks=0.1*i + seq(1, 6, 0.5),
        prob=T, plot=F)$intensities, rep(5,10))
```

Pages 180–1 and 185–6 use Trellis plots.

### Chapter 6

The default contrasts are different in R, so use

```
options(contrasts=c("contr.helmert", "contr.poly"))
```

for this chapter.

Function `xypplot` (page 192) is a Trellis function.

Functions `predict.gam` (page 209) and `fac.design` (page 213) are not implemented in R.

## Chapter 7

Powers need to be protected in formulae as in

```
anova(update(budworm.lg, . ~ . + sex*I(ldose^2)), test="Chisq")
```

on page 232.

Summary tables from `glm` fits have a column of significance of the  $t$  ratio, called `z` value. Given the Hauck-Donner effect, view these with caution.

## Chapter 8

Only a few robust functions are implemented in R. Those missing include `location.m`, `scale.a`, `scale.tau` (page 253), `l1fit` (page 257), `rreg` (page 259), `family.robust` (page 259), `lmsreg`, `ltsreg` (page 262) and `cov.mve` (page 266). Functions `lmsreg`, `ltsreg` and `cov.mve` are provided as wrappers in package `lqs`.

## Chapter 9

There is a function `nls` in package `nls` available from CRAN (and which depends on package `lme`), but as yet only a few support functions. There is a `nlmin`-like function called `nlm`, but no other optimization functions (as yet).

## Chapter 10

Function `aov` allows Error terms, and there is a package `lme` on CRAN which can be used for the `lme` examples: see the R script `ch10` for the syntax needed.

There are no functions `is.random`, `raov`, `varcomp` nor `nlme`<sup>2</sup> in R.

---

<sup>2</sup> This exists in package `lme` but is not operational.

## Chapter 11

Only some of the smoothing functions of page 324–6 exist in base R. The spline functions `bs` and `ns` are in package `splines`, and `smooth.spline`, `ksmooth` and `supsmu` in package `modreg`.

There are no functions `gam` and `ppreg` (but see function `ppr` described in our on-line Statistics Complements, which is in package `modreg`).

Functions `ace` and `avas` are in package `acepack`; `avas` has argument `lin` not `linear` (page 337).

## Chapter 12

Package `survival4` is needed for this chapter, and that requires package `splines`.

Dataset `VA` (page 363) is supplied in package `MASS`.

Function `scatter.smooth` (pages 264 and 376) is in package `modreg`.

Assigning contrasts to terms does not work with the current `survival4`, so on page 376 we could use

```
c.age <- factor(as.character(c.age), levels = c("31-40", "0-15",  
      "16-30", "41-50", "51-60", "61+"))
```

rather than assigning contrasts.

The expected survival examples can be done using packages `ratetables` and `date`.

## Chapter 13

Package `mva` is needed.

Data set `swiss.x` can be obtained as `swiss.x <- as.matrix(swiss[, -1])`.

There are no functions `brush` (page 382), `mstree` (page 387), `cutree`, `mclust`, `mreloc` nor `factanal`.

Eigenvectors are only defined up to a change in sign, so the plots from principal components, MDS and `lda` may be reflected about either or both of the axes.

## Chapter 14

Package `tree` is needed. A port of `rpart` as described in the on-line Statistics Complements is available on CRAN.

## Chapter 15

Almost all these examples can be done with library `ts` in R 0.65.0 and later, although the syntax of the commands is often somewhat different. See the script `ch15` for details of the changes.

## Chapter 16

The only difference is the absence of Trellis functions, and a script based on the code in our first edition is supplied both with the `spatial` package and in the `scripts` directory.

## Chapter 17

Log scales are handled differently for `contour` plots (page 488 and elsewhere).

Assignments with `assign` need to be to the `.GlobalEnv` environment not `frame 1` (page 495). There are alternative ways to handle this in R.

## References

- Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The NEW S Language*. New York: Chapman and Hall. (Formerly Monterey: Wadsworth and Brooks/Cole.). [7]
- Chambers, J. M. and Hastie, T. J. eds (1992) *Statistical Models in S*. New York: Chapman and Hall. (Formerly Monterey: Wadsworth and Brooks/Cole.). [7]
- Ihaka, R. and Gentleman, R. (1996) R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* **5**(3), 299–314. [2, 18, 19, 20]
- Murrell, P. and Ihaka, R. (1999) An approach to providing mathematical annotation in plots. *Journal of Computational and Graphical Statistics* **XX**, xxx–xxx. [13]
- Wichmann, B. A. and Hill, I. D. (1982) Algorithm AS183. an efficient and portable pseudo-random number generator. *Applied Statistics* **31**, 188–190. (Correction **33**, 123.). [9]

# Index

Entries in this font are names of S objects.

- +, 4
- .First, 11
- .First.lib, 24
- .GlobalEnv, 18, 33
- .Last, 11
- .Library, 24
- .RData, 2
- .Random.seed, 9
- .Rprofile, 11
- .lib.loc, 24
- <<-, 19
- >, 4
- [[, 5
  
- ace, 1, 32
- aov, 31
- apropos, 10
- assign, 18, 33
- assignment, 19
- avas, 1, 32
  
- batch operation, 11
- body, 27
- browser
  - default, 10
- browser, 15–17
- brush, 32
- bs, 32
- bwplot, 30
- by, 30
  
- case.names, 10
- cdf.compare, 30
- cex, 14
- chol2inv, 10
- choose, 10
- colnames, 10
- colours
  - specifying, 13
- command history, 4
- command-line editing, 4
- cons cells, 3
- contour, 33
- contourplot, 29
- cov.mve, 31
- CRAN, 1
- crosstabs, 30
- cutree, 32
  
- data, 8, 28, 29
- datasets
  - loading, 8
- debug, 17
- debugger, 15
- debugging, 15
- digamma, 10
- dos, 17
- drop.terms, 10
- dyn.load, 20
- dynamic loading, 20
  
- editing functions, 15
- environment, 19
- environments, 18
- errors
  - finding, 15
- ESS, 4, 17
- eval, 18, 27
- evalq, 18
- exists, 18
- expression, 13
  
- fac.design, 31
- factanal, 32
- false, 7
- family, 27
- font, 14
- formals, 27
- formulae, 9
- frames, 9, 18

- Free, 21
- functions
  - calling C, 20
  - calling FORTRAN, 20
  - editing, 15
- gam, 32
- garbage collection, 3
- get, 18
- gl, 10
- glm, 27
- gray, 13
- gsub, 10
- heap, 3
- help
  - on-line, 5
  - preparing, 17
  - printing, 5
- help, 5
- help.start, 5, 10
- hist, 30
- hist.default, 20
- hsv, 13
- inspect, 15
- integrate, 30
- interp, 1
- interrupt, 4
- IQR, 10
- iris, 29
- iris3, 29
- is.matrix, 8
- is.R, 10
- isR, 27
- ksmooth, 32
- l1fit, 31
- lazy evaluation, 20
- levelplot, 29
- libraries
  - converting, 26
- library, 24
- library.dynam, 20
- lists, 8
  - removing components, 8
- lme, 31
- lmsreg, 31
- loading
  - dynamic, 20
  - location.m, 31
  - loess, 29
  - ltsreg, 29, 31
  - lung.deaths, 30
- mailing lists, 2
- mat.or.vec, 10
- matrix
  - indexing, 8
  - testing for, 8
- memory management, 2
  - garbage collection, 3
- merge, 30
- mreloc, 32
- mstree, 32
- NCOL, 9
- nlevels, 10
- nlm, 31
- nlmin, 31
- nls, 31
- NROW, 9
- ns, 32
- objects
  - storage mode, 20
- operating system
  - calls to, 17
- options, 10
- order, 29
- package
  - acepack, 1, 32
  - akima, 1
  - base, 24
  - boot, 1
  - bootstrap, 1
  - class, 1
  - cluster, 1
  - ctest, 1, 30
  - date, 32
  - eda, 24
  - integrate, 30
  - KernSmooth, 1
  - lme, 31
  - locfit, 1
  - logspline, 1
  - lqs, 24, 29, 31
  - MASS, 1, 10, 23, 30, 32

- Matrix, 29
- modreg, 24, 29, 32
- mva, 24, 29, 32
- nls, 31
- nnet, 1
- ppr, 1
- ratetables, 32
- rpart, 1, 32
- sm, 1
- spatial, 1, 33
- splines, 1, 32
- stepfun, 24
- survival4, 1, 32
- tree, 1, 32
- ts, 33
- packages, 1
  - creating, 24
  - private libraries, 24
  - using, 23
- palette, 13
- paper size, 10
- par, 13, 30
- plots
  - enhancing, 12
- ppr, 32
- ppreg, 32
- predict.gam, 31
- print command, 10
- profile files, 5, 10
- prompt, 17
- q, 4
- qqmath, 30
- R, 1
- R batch operation, 11
- R-core, 2
- R history, 2
- R memory management, 2
- R profiles, 5, 10
- random numbers, 9
- rgb, 13
- RLIBS, 24
- RNGkind, 9
- robust, 31
- rownames, 10
- rreg, 29, 31
- scale.a, 31
- scale.tau, 31
- scatter.smooth, 32
- set.seed, 9
- single, 20
- smooth.spline, 32
- sort.list, 29
- split.screen, 30
- starting
  - under Unix, 3
  - under Windows, 6
- stem, 30
- storage mode, 20
- sub, 10
- subplot, 30
- substitute, 18, 20
- supsmu, 32
- swiss, 29, 30
- swiss.df, 30
- swiss.fertility, 29
- swiss.x, 29, 32
- sys.frame, 18
- sys.parent, 18
- system, 17
- traceback, 15
- trigamma, 10
- true, 7
- undebug, 17
- uniroot, 27
- Unix, 1, 3, 5, 6, 10, 11, 17, 20, 24–26, 28
- unix, 17
- usa, 29
- VA, 32
- var, 5
- variable.names, 10
- vectors
  - indexing, 8
- win3, 17
- Windows, 1, 6, 11, 20, 22, 25
- wireframe, 29
- workspace, 2
  - saving, 2
  - setting size of, 3
- XERROR, 22
- xyplot, 31