‘R’ Complements to

Modern Applied Statistics with S-Plus

Third edition

by

W. N. Venables and B. D. Ripley

Springer (1999). ISBN 0-387-98825-4

14 December 2001

These complements have been produced to supplement the third edition of MASS. They will be updated from time to time. The definitive source is http://www.stats.ox.ac.uk/pub/MASS3/.

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 earlier versions of R compatible with the S code we use.

There are separate Complements documents for S statistical methods available from

`http://www.stats.ox.ac.uk/pub/MASS3/`.

Some of these will be of little interest to R users; perhaps most relevant are Sections 5.6, 7.6, 9.1, 11.3, 12.1 and 12.6 which describe methods in packages by ourselves and others which are available for R.

R is discussed in much greater depth in Venables & Ripley (2000).

Contents

Chapter 1

Introduction

R is a Open Source 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, for 32-bit versions of Windows and for the Macintosh.[1] 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://cran.r-project.org/mirrors.html`

From a user's perspective R is very similar to S; indeed until recently the only real documentation of the R language was via lists of differences from S. (An R language manual was released with version 1.2.0, but is incomplete.) The R system is still under 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 some 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 graphics (a similar system called lattice is under development)
brush and spin plots (but XGobi or GGobi can be used)

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 10 (trees) you will need packages `rpart` and `tree`. To use Chapter 12 (survival analysis) you will need the package `survival`. Other packages that are needed in places are `nlme` (Chapters 6, 8 and 13), `acepack` (functions `ace` and `avas` in Chapter 9), `mgcv` (for `gam` in Chapter 9), `akima` (function `interp` used in Chapter 14) and `cluster`. Further, some of the libraries used in the text and in the on-line statistical complements are also available for R, including

`KernSmooth`, `boot`, `locfit`, `logspline`, `sm`.

Many of these (our libraries `MASS`, `nnet`, `spatial` and `class`, `KernSmooth`, `boot`, `mgcv`, `nlme`, `rpart`, `survival`) are 'recommended packages' which should be available in all installations of R.

[1] under both the classic MacOS and MacOS X.

We will assume that you are using a version of R not earlier than 1.2.0, 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[2] 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[3].

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.

R version 1.0.0 was released on 29 February 2000. This was the first non-beta version and thereafter the pace of change in R has slowed down, with major releases about twice a year.

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).

[2] including BDR since 1999
[3] See the CRAN web pages for how to subscribe.

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()`.

Prior to version 1.2.0, R workspaces were static. They are now dynamically sized, but some understanding of the workspace is needed to use R well. The workspace has two components, the 'cons cells' and the 'heap'. R uses garbage collection to manage the memory, so when it runs out of free space, it first removes all unused objects in the workspace, and then if needed allocates some more space. During the process, the 'limits' can be moved up or down according to current usage (but move very slowly unless raised to accommodate a large object). You can see the current state of the workspace by triggering a garbage collection with `gc()` which will tell you the current usage and the current 'limits'.

You can set the minimum and maximum values for the number of cons cells by `--min-nsize` and `--max-nsize` and for the vector heap by `--min-vsize` and `--max-vsize`, but you will need to do so only rarely. If you have lots of RAM available, increasing the minima will improve performance slightly. If you have limited RAM or virtual memory, setting the maxima will cause R to stop a task rather than page excessively or fill up the virtual memory.

See Appendix A for details of how to use specific R implementations.

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 contains some deliberate differences, but unintentional ones are still being found.

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.

R regards `TRUE` and `FALSE` as the logical values but allows `T` and `F` as abbreviations[1] that are expanded on printing.

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

S has the converse convention.

In R a vector with names is a vector according to `is.vector`: it is not in S.

The set of 'roman letters' (page 13) allowed in R object names is locale-dependent. In the `C` locale it is upper- and lower-case `a ... z`, but in other locales it may include accented and other letters. (Whether locale support works is platform-dependent.)

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 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.

[1] 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.

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

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[, FALSE]` 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. As from R 0.99.0 it is a multiply-with-carry RNG suggested by Marsaglia in 1997 with a period of more than 2^{60}. Even higher-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, which includes plugging in a user’s own generator.

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.

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 are represented differently: `%in%` is not used in R for model formulae (but is used for what is `is.element` in both S and R).

The handling of `-` mentioned on page 161 is different in R: `a*b - b` *is* equal to `a + a:b`.

The defaults for `na.action` arguments in forming model frames are identical to those for S, *except* that `options(na.action = na.omit)` is part of the standard R startup sequence, and that option is unset in S-PLUS. The net effect is that S users will surprised to find rows with missing data silently omitted in R.

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.
`rownames` `rownames<-` `colnames` `colnames<-`	give or set the first and second component of the `dimnames` of a matrix, array or data frame. (A missed opportunity here: these do not work for vectors.)
`apropos`	a `find`-like function using regular expressions.
`case.names`	extract the names of cases (observations) from a fit.
`chol2inv`	inverse from Choleski decomposition.
`choose`	binomial coefficients (as in S-PLUS 4.x and 5.x).
`digamma` `trigamma`	digamma and trigamma functions (and also other derivatives).
`drop.terms`	drop terms in a `terms` object.
`gl`	a ‘generate levels’ function based on `%GL` in GLIM.
`gsub`, `sub`	substitute regular expressions in strings.
`IQR`	Inter-Quartile Range, as in library `MASS` for S-PLUS.
`is.R`	R function which checks if the code is being run in R. See page 20 of these complements or its help page for how to use it.
`mat.or.vec`	create a matrix or vector.
`nlevels`	the number of levels of a factor.
`variable.names`	extract the names of variables from a fit.

Chapter 3

Graphics

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.

- finctions `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 (2000), and more details and many examples can be obtained by `?plotmath`.

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[1] 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())
```

[1] 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

4.2 More on character strings

R has several functions of searching and matching with regular expressions. Function `grep` works with `egrep`–like expressions. Functions `sub` and `gsub` have syntax

```
sub(pattern, replacement, x, ignore.case=FALSE, extended=TRUE)
gsub(pattern, replacement, x, ignore.case=FALSE, extended=TRUE)
```

(and `grep` also has the last two arguments). Both replace `pattern` by `replacement` in each element of the character vector `x`, but `sub` replaces on the first occurrence. Thus conceptually they apply the `sed` / `Perl` commands `s/pattern/replacement/` and `s/pattern/replacement/g` to the strings in `x`. The `ignore.case` argument should be self-evident (in `Perl` it is qualifier `i`); if `extended` is false the regular expressions are `grep`—rather than `egrep`—like.

Function `regexpr` is available. R now compiles in the GNU regex library, so the interpretation of regular expressions is consistent across all platforms.

Appendix A

Getting Started

A.4 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
   ```

   You will see a banner[1] similar to

   ```
   R : Copyright 2001, The R Development Core Team
   Version 1.3.1  (2001-08-31)

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

   R is a collaborative project with many contributors.
   Type ‘contributors()’ for more information.

   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]
   ```

[1] 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.[2]

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

`http://ess.stat.wisc.edu`

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. Note that R's idea of the current expression may be differ from S's, and may be smaller.

[2] You will probably find `man readline` tells you about the many options that are available.

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 `less`; 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)
set.seed(123)
```

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`.

A.5 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, ME, NT4, 2000 and XP, and is available from CRAN in directory `bin/windows/base`. 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 `--max-vsize` 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 12) in an almost identical fashion to under Unix; see recent versions of ESS 5.1.x for details.

There is a Windows-specific flag `--max-mem-size` to control overall memory usage. It defaults to the smaller of 256Mb and the amount of physical RAM available to Windows.

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

There are several formats of help available, not all of which need be installed; see `?help` or the `README` 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` or `Ctrl-C` in `Rterm.exe`: `Ctrl-C` is used for copying in the GUI version.
- Command-line editing is always available, but is simpler than the `readline`-based editing on Unix. For `Rgui.exe`, the menu item `Help | Console` will give details and for `Rterm.exe` see the file `README.Rterm`.
- Using `help.start()` does not automatically send help requests to the `browser()`: use `options(htmlhelp = TRUE)` 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.

There are also some additional features in the Windows version, notably the ability to save and replay graphics commands, and the `bmp` bitmap graphics device. See the `README` or `rw-FAQ` for more details.

A.6 Customizing your R environment

There are many fewer options that can be set by `options` than in the table on page 65 of the second edition. 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` has a limited effect in R; commands are normally echoed if input is from a file unless `--quiet` or `--slave` was used, and are never echoed if input is from a terminal (or terminal-like connection). Option `echo` only affects input from a file.

There are some further options specific to R:

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

Some of these can also be set in environmental variables, for example `R_PRINTCMD` and `R_PAPERSIZE`. The variable `R_LIBS` controls the default search path for packages (see page 17 of these complements).

On all versions, environment variables to be used with R can be set in the file `~/.Renviron` as name = value pairs, for example

```
R_LIBS=/ext/R/library
```

We have already mentioned the use of `.Rprofile` on page 13. It is also possible to have a system-wide profile file which is read before the user's file (if any). The site profile should be stored in `${R_HOME}/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.

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/library’:

KernSmooth      Functions for kernel smoothing for Wand &Jones (1995)
MASS            Main Library of Venables and Ripley’s MASS
acepack         ace() and avas() for selecting regression
                transformations
akima           Interpolation of irregularly spaced data
boot            Bootstrap R (S-Plus) Functions (Canty)
class           Functions for classification
cluster         Functions for clustering
date            Functions for handling dates
integrate       numerical integration
locfit          Local Regression, Likelihood and Density Estimtion.
logspline       Logspline density estimation
nlme            Linear and nonlinear mixed effects models
nnet            Feed-forward neural networks and multinomial log-linear
                models
rpart           Recursive partitioning
sm              kernel smoothing methods: Bowman & Azzalini (1997)
spatial         functions for kriging and point pattern analysis
survival        Survival analysis, including penalised likelihood.
tree            Classification and regression trees

Packages in library ‘/ext/R/current/lib/R/library’:

base            The R base package
ctest           Classical Tests
eda             Exploratory Data Analysis
lqs             Resistant Regression and Covariance Estimation
modreg          Modern regression: Smoothing and Local Methods
mva             Classical Multivariate Analysis
nls             Nonlinear regression
stepfun         Step Functions, including Empirical Distributions
splines         Regression Spline Functions and Classes
tcltk           Interface to Tcl/Tk
ts              Time series functions
```

Some of these are standard (the second group) and the others have been installed as extensions to R. To find out what is in a package use the `help` argument,

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

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

Packages in places other than the standard library may be used by specifying the argument `lib.loc` to `library`[1] 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 `R_LIBS` and `.Library`, and setting `R_LIBS` is the preferred way to change the library search path. You will not find the help for functions in packages in private libraries unless `R_LIBS` (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:survival")
```

If you have Internet access and the utilities they need, the functions `install.packages` and `update.packages` will download packages from a CRAN node and install/update them.

The function `installed.packages` gives version information on the installed packages.

C.2 Creating a package

A package in R contains files `DESCRIPTION`, `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=TRUE)` (`.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)
```

[1] Note that unlike S-PLUS, the system library is not searched when `lib.loc` is specified

Note the different orders of the arguments!

For further details see the manual *Writing R Extensions*. Do remember to write the documentation files such as `DESCRIPTION` (see page 19)!

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

```
R CMD check pkgname
```

on the source tree will provide a check of this. The utility

```
R CMD build pkgname
```

will build a `tar` file of the package after running a number of checks of completeness. Under Windows, replace `R CMD` by `Rcmd`.

C.3 Installing R packages

This is straightforward under Unix. A package is usually distributed as a file with extension `.tar.gz` or `.tgz`. For the first just use

```
R CMD INSTALL [-l path/to/library] name.tar.gz
```

which unpacks the archive in a temporary location and then installs it, by default in the system library and optionally (using the the part in `[ ]`) in another library.

Alternatively, first unpack the source tree 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 CMD INSTALL [-l path/to/library] libname
```

Installation is even easier under Windows; just use the Packages menu. Alternatively, change directory to the library directory under `R_HOME` and unzip the pre-packaged zip file. To install a package from source on Windows see the instructions in the distribution, in particular those in the *R for Windows FAQ*.

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 `csh`, 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 `DESCRIPTION` file along the lines of

```
Package: mytest
Version: 0.2-1
Title: A title
Author: An author <author@dept.domain.dom>
Maintainer: <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 CMD INSTALL .
```

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

- Missing functions (e.g. `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 some options for the `type` argument, and `uniroot` has fewer arguments under R).
- Applying `is.vector` to a vector with names gives true in R and false in S.
- Applying `is.matrix` to a data frame gives false in R and true in S.
- `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()` or to make use of R's `eval.parent`.
- Plot labels and titles may need to be evaluated early in the function 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 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`.
- In S the scope of the index variable of a `for` loop is the body of the loop: in R it is the environment containing the `for` loop. More generally, braced expressions generate a separate frame in S, but have no special handling in R.

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

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

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`, but there is limited support for passing `float` variables to C and `REAL` variables for FORTRAN. Use `as.single` and `single` or `storage.mode<-`: the internal variables in R are still double-precision, but single-precision copies are passed to and from the compiled code.

R does have fairly comprehensive support for C++: see *Writing R Extensions* for hints and examples.

Create a directory `src` and move the C, 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 CMD SHLIB -o mytest.so *.{c,f,cc,C,cpp}
```

It is rarely necessary to write a `Makefile`, but one can be included in the `src` directory.

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).

The `PACKAGE = "mypkg"` argument to `.C` and `.Fortran` can be used to confine the search for symbols to your library: its use is highly recommended as protection against name clashes between packages.

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`.[1]

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

Another theme (for now) is the absence of Trellis graphics. The current development version of `lattice` does not run some of our examples correctly.

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

Chapter 1

On page 6, `trellis.device()` should be omitted.

On page 6, `contourplot`, `wireframe` and `levelplot` are Trellis functions which are missing.

On page 8, replace `splom(~ hills)` by `pairs(hills)`.

Chapter 3

There is no function `brush` (page 60). `contourplot` (page 62) is a Trellis function which although it exists in `lattice` does not work.

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 72.

[1] with numbers entered to a higher precision.

None of Sections 3.5 and 3.6 is relevant (yet). There is a package `lattice` under development (in the `Devel` area on CRAN) that is a reasonable approximation to Trellis.

Chapter 4

Dataset `iris` is called `iris3` (pp. 104–5).

Functions `crosstabs` (page 105) does not exist in R, but there is a similar function `xtabs`.

Chapter 5

Function `qqmath` (page 115) is a Trellis function.

Argument `scale` to `stem` acts differently: we could use `stem(abbey, scale=0.4)` (page 121). The `stem` plots come out rather differently as the outliers are not excluded in R.

Function `bwplot` (page 122) is a Trellis function.

The functions of Section 5.4 are in package `ctest`. However, there is no `cdf.compare` (page 126).

Page 140 uses Trellis plots.

`bootstrap` (page 144) is an S-PLUS function.

Chapter 6

The default contrasts are different in R, so use

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

for this chapter.

Function `xyplot` (page 150) is a Trellis function.

Functions `predict.gam` (page 166), `lmRobMM` (page 172), `fac.design` (page 179), `multicomp` (pages 190–2), `raov` (page 193), `is.random` and `varcomp` (page 194) are not implemented in R. Package `nlme` on CRAN provides `lme`.

Chapter 7

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

Function `nls` is in standard package `nls`. Most of the optimization examples can be done using `nlm` or `optim`, but symbolic differentiation using `deriv` does not work for our examples. See the script `ch08.R` for details.

Function `nlme` is in package `nlme`.

Chapter 9

Only some of the smoothing functions of page 282–3 exist in base R. The spline functions `bs` and `ns` are in standard package `splines`, and `smooth.spline`, `ksmooth` and `supsmu` in standard package `modreg`.

There is no function `gam` in base R, but something very similar to these examples can be done by function `gam` in package `mgcv`: see the script `ch09.R`.

Function `ppr` is in standard package `modreg`.

Functions `ace` and `avas` are in package `acepack`; `avas` has argument `lin` not `linear` (page 295).

Chapter 10

Packages `rpart` and `tree` are needed.

Chapter 11

Standard package `mva` is needed.

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

There are no functions `brush` (page 330), `mclust` nor `mreloc`.

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 12

Package `survival` is needed for this chapter.

Function `censorReg` (pages 379–80) does not exist in R.

Dataset `VA` (page 387) is supplied in package `MASS`.

Function `scatter.smooth` (pages 388 and 397) is in the standard package `modreg`.

Chapter 13

Almost all these examples can be done with the standard package `ts`, although the syntax of the commands is often somewhat different. See the script `ch13.R` for details of the changes.

Chapter 14

None.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The NEW S Language*. New York: Chapman & Hall. (Formerly Monterey: Wadsworth and Brooks/Cole.). [4]

Chambers, J. M. and Hastie, T. J. eds (1992) *Statistical Models in S*. New York: Chapman & Hall. (Formerly Monterey: Wadsworth and Brooks/Cole.). [4]

Ihaka, R. and Gentleman, R. (1996) R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* **5**, 299–314. [2]

Murrell, P. and Ihaka, R. (2000) An approach to providing mathematical annotation in plots. *Journal of Computational and Graphical Statistics* **9**, 582–599. [8]

Venables, W. N. and Ripley, B. D. (2000) *S Programming*. New York: Springer-Verlag. [i]

Index

Entries in `this font` are names of S objects.