Appendix A A sample session

The following session is intended to introduce to you some features of the R environment by using them. Many features of the system will be unfamiliar and puzzling at first, but this puzzlement will soon disappear.

Login, start your windowing system.

`$ R` Start R as appropriate for your platform.

The R program begins, with a banner.

(Within R, the prompt on the left hand side will not be shown to avoid confusion.)

`help.start()`
Start the HTML interface to on-line help (using a web browser available at your machine). You should briefly explore the features of this facility with the mouse.

Iconify the help window and move on to the next part.

```
x <- rnorm(50)
y <- rnorm(x)
```
Generate two pseudo-random normal vectors of x- and y-coordinates.

`plot(x, y)`
Plot the points in the plane. A graphics window will appear automatically.

`ls()` See which R objects are now in the R workspace.

`rm(x, y)` Remove objects no longer needed. (Clean up).

`x <- 1:20` Make $x = (1, 2, \ldots, 20)$.

`w <- 1 + sqrt(x)/2`
A 'weight' vector of standard deviations.

```
dummy <- data.frame(x=x, y= x + rnorm(x)*w)
dummy
```
Make a *data frame* of two columns, x and y, and look at it.

```
fm <- lm(y ~ x, data=dummy)
summary(fm)
```
Fit a simple linear regression of y on x and look at the analysis.

```
fm1 <- lm(y ~ x, data=dummy, weight=1/w^2)
summary(fm1)
```
Since we know the standard deviations, we can do a weighted regression.

`attach(dummy)`
Make the columns in the data frame visible as variables.

`lrf <- lowess(x, y)`
Make a nonparametric local regression function.

`plot(x, y)`
Standard point plot.

`lines(x, lrf$y)`
Add in the local regression.

`abline(0, 1, lty=3)`
The true regression line: (intercept 0, slope 1).

`abline(coef(fm))`
Unweighted regression line.

```
abline(coef(fm1), col = "red")
```
Weighted regression line.

```
detach()
```
Remove data frame from the search path.

```
plot(fitted(fm), resid(fm),
     xlab="Fitted values",
     ylab="Residuals",
     main="Residuals vs Fitted")
```
A standard regression diagnostic plot to check for heteroscedasticity. Can you see it?

```
qqnorm(resid(fm), main="Residuals Rankit Plot")
```
A normal scores plot to check for skewness, kurtosis and outliers. (Not very useful here.)

```
rm(fm, fm1, lrf, x, dummy)
```
Clean up again.

The next section will look at data from the classical experiment of Michaelson and Morley to measure the speed of light. This dataset is available in the `morley` object, but we will read it to illustrate the `read.table` function.

```
filepath <- system.file("data", "morley.tab" , package="datasets")
filepath
```
Get the path to the data file.

```
file.show(filepath)
```
Optional. Look at the file.

```
mm <- read.table(filepath)
mm
```
Read in the Michaelson and Morley data as a data frame, and look at it. There are five experiments (column `Expt`) and each has 20 runs (column `Run`) and `sl` is the recorded speed of light, suitably coded.

```
mm$Expt <- factor(mm$Expt)
mm$Run <- factor(mm$Run)
```
Change `Expt` and `Run` into factors.

```
attach(mm)
```
Make the data frame visible at position 3 (the default).

```
plot(Expt, Speed, main="Speed of Light Data", xlab="Experiment No.")
```
Compare the five experiments with simple boxplots.

```
fm <- aov(Speed ~ Run + Expt, data=mm)
summary(fm)
```
Analyze as a randomized block, with 'runs' and 'experiments' as factors.

```
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
```
Fit the sub-model omitting 'runs', and compare using a formal analysis of variance.

```
detach()
rm(fm, fm0)
```
Clean up before moving on.

We now look at some more graphical features: contour and image plots.

```
x <- seq(-pi, pi, len=50)
y <- x
```
x is a vector of 50 equally spaced values in $-\pi \le x \le \pi$. y is the same.

`f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))`
: f is a square matrix, with rows and columns indexed by x and y respectively, of values of the function $\cos(y)/(1 + x^2)$.

```
oldpar <- par(no.readonly = TRUE)
par(pty="s")
```
: Save the plotting parameters and set the plotting region to "square".

```
contour(x, y, f)
contour(x, y, f, nlevels=15, add=TRUE)
```
: Make a contour map of f; add in more lines for more detail.

`fa <- (f-t(f))/2`
: `fa` is the "asymmetric part" of f. (`t()` is transpose).

`contour(x, y, fa, nlevels=15)`
: Make a contour plot, ...

`par(oldpar)`
: ... and restore the old graphics parameters.

```
image(x, y, f)
image(x, y, fa)
```
: Make some high density image plots, (of which you can get hardcopies if you wish), ...

`objects(); rm(x, y, f, fa)`
: ... and clean up before moving on.

R can do complex arithmetic, also.

```
th <- seq(-pi, pi, len=100)
z <- exp(1i*th)
```
: `1i` is used for the complex number i.

```
par(pty="s")
plot(z, type="l")
```
: Plotting complex arguments means plot imaginary versus real parts. This should be a circle.

`w <- rnorm(100) + rnorm(100)*1i`
: Suppose we want to sample points within the unit circle. One method would be to take complex numbers with standard normal real and imaginary parts ...

`w <- ifelse(Mod(w) > 1, 1/w, w)`
: ... and to map any outside the circle onto their reciprocal.

```
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+",xlab="x", ylab="y")
lines(z)
```
: All points are inside the unit circle, but the distribution is not uniform.

```
w <- sqrt(runif(100))*exp(2*pi*runif(100)*1i)
plot(w, xlim=c(-1,1), ylim=c(-1,1), pch="+", xlab="x", ylab="y")
lines(z)
```
: The second method uses the uniform distribution. The points should now look more evenly spaced over the disc.

`rm(th, w, z)`
: Clean up again.

`q()`
: Quit the R program. You will be asked if you want to save the R workspace, and for an exploratory session like this, you probably do not want to save it.

Appendix B Invoking R

B.1 Invoking R from the command line

When working in UNIX or at a command line in Windows, the command '`R`' can be used both for starting the main R program in the form

`R` [*options*] [`<`*infile*] [`>`*outfile*],

or, via the `R CMD` interface, as a wrapper to various R tools (e.g., for processing files in R documentation format or manipulating add-on packages) which are not intended to be called "directly".

Under UNIX you do need to ensure that either the environment variable `TMPDIR` is unset or it points to a valid place to create temporary files and directories.

Most options control what happens at the beginning and at the end of an R session. The startup mechanism is as follows (see also the on-line help for topic '`Startup`' for more information, and the section below for some Windows-specific details).

- Unless '`--no-environ`' was given, R searches for user and site files to process for setting environment variables. The name of the site file is the one pointed to by the environment variable `R_ENVIRON`; if this is unset, '`$R_HOME/etc/Renviron.site`' is used (if it exists). The user file searched for is '`.Renviron`' in the current or in the user's home directory (in that order). These files should contain lines of the form '*`name=value`*'. (See `help(Startup)` for a precise description.) Variables you might want to set include `R_PAPERSIZE` (the default paper size), `R_PRINTCMD` (the default print command) and `R_LIBS` (specifies the list of R library trees searched for add-on packages).
- Then R searches for the site-wide startup profile unless the command line option '`--no-site-file`' was given. The name of this file is taken from the value of the `R_PROFILE` environment variable. If that variable is unset, the default '`$R_HOME/etc/Rprofile.site`' is used if this exists.
- Then, unless '`--no-init-file`' was given, R searches for a file called '`.Rprofile`' in the current directory or in the user's home directory (in that order) and sources it.
- It also loads a saved image from '`.RData`' if there is one (unless '`--no-restore`' or '`--no-restore-data`' was specified).
- Finally, if a function `.First` exists, it is executed. This function (as well as `.Last` which is executed at the end of the R session) can be defined in the appropriate startup profiles, or reside in '`.RData`'.

In addition, there are options for controlling the memory available to the R process (see the on-line help for topic '`Memory`' for more information). Users will not normally need to use these unless they are trying to limit the amount of memory used by R.

R accepts the following command-line options.

'`--help`'
'`-h`' Print short help message to standard output and exit successfully.

'`--version`'
Print version information to standard output and exit successfully.

'`--encoding=`*`enc`*'
Specify the encoding to be assumed for input from the console or `stdin`. This needs to be an encoding known to `iconv`: see its help page.

'`RHOME`' Print the path to the R "home directory" to standard output and exit successfully. Apart from the front-end shell script and the man page, R installation puts everything (executables, packages, etc.) into this directory.

‘`--save`’
‘`--no-save`’
: Control whether data sets should be saved or not at the end of the R session. If neither is given in an interactive session, the user is asked for the desired behavior when ending the session with `q()`; in non-interactive use one of these must be specified.

‘`--no-environ`’
: Do not read any user file to set environment variables.

‘`--no-site-file`’
: Do not read the site-wide profile at startup.

‘`--no-init-file`’
: Do not read the user’s profile at startup.

‘`--restore`’
‘`--no-restore`’
‘`--no-restore-data`’
: Control whether saved images (file ‘`.RData`’ in the directory where R was started) should be restored at startup or not. The default is to restore. (‘`--no-restore`’ implies all the specific ‘`--no-restore-*`’ options.)

‘`--no-restore-history`’
: Control whether the history file (normally file ‘`.Rhistory`’ in the directory where R was started, but can be set by the environment variable `R_HISTFILE`) should be restored at startup or not. The default is to restore.

‘`--vanilla`’
: Combine ‘`--no-save`’, ‘`--no-environ`’ ‘`--no-site-file`’, ‘`--no-init-file`’ and ‘`--no-restore`’.

‘`--no-readline`’
: (UNIX only) Turn off command-line editing via **readline**. This is useful when running R from within Emacs using the ESS (“Emacs Speaks Statistics”) package. See Appendix C [The command-line editor], page 83, for more information.

‘`--ess`’ (Windows only) Set `Rterm` up for use by `R-inferior-mode` in ESS.

‘`--min-vsize=`*N*’
‘`--max-vsize=`*N*’
: Specify the minimum or maximum amount of memory used for variable size objects by setting the “vector heap” size to *N* bytes. Here, *N* must either be an integer or an integer ending with ‘`G`’, ‘`M`’, ‘`K`’, or ‘`k`’, meaning ‘Giga’ (2^30), ‘Mega’ (2^20), (computer) ‘Kilo’ (2^10), or regular ‘kilo’ (1000).

‘`--min-nsize=`*N*’
‘`--max-nsize=`*N*’
: Specify the amount of memory used for fixed size objects by setting the number of “cons cells” to *N*. See the previous option for details on *N*. A cons cell takes 28 bytes on a 32-bit machine, and usually 56 bytes on a 64-bit machine.

‘`--max-ppsize=`*N*’
: Specify the maximum size of the pointer protection stack as *N* locations. This defaults to 10000, but can be increased to allow large and complicated calculations to be done. Currently the maximum value accepted is 100000.

‘`--max-mem-size=`*N*’
: (Windows only) Specify a limit for the amount of memory to be used both for R objects and working areas. This is set by default to the smaller of 1024Mb and the amount of physical RAM in the machine, and must be at least 16Mb.

‘`--quiet`’
‘`--silent`’
‘`-q`’
: Do not print out the initial copyright and welcome messages.

‘`--slave`’
: Make R run as quietly as possible. This option is intended to support programs which use R to compute results for them.

‘`--verbose`’
: Print more information about progress, and in particular set R's option `verbose` to `TRUE`. R code uses this option to control the printing of diagnostic messages.

‘`--debugger=`*name*’
‘`-d` *name*’
: (UNIX only) Run R through debugger *name*. Note that in this case, further command line options are disregarded, and should instead be given when starting the R executable from inside the debugger.

‘`--gui=`*type*’
‘`-g` *type*’
: (UNIX only) Use *type* as graphical user interface (note that this also includes interactive graphics). Currently, possible values for *type* are ‘`X11`’ (the default), provided that ‘`Tcl/Tk`’ support is available, ‘`Tk`’ and ‘`gnome`’ provided that package **gnomeGUI** is installed.

‘`--args`’
: This flag does nothing except cause the rest of the command line to be skipped: this can be useful to retrieve values from it with `commandArgs()`.

Note that input and output can be redirected in the usual way (using ‘`<`’ and ‘`>`’). Warning and error messages are sent to the error channel (`stderr`) except on Windows 9X/ME.

The command `R CMD` allows the invocation of various tools which are useful in conjunction with R, but not intended to be called “directly”. The general form is

```
R CMD command args
```

where *command* is the name of the tool and *args* the arguments passed on to it.

Currently, the following tools are available.

`BATCH`
: Run R in batch mode.

`COMPILE`
: (UNIX only) Compile files for use with R.

`SHLIB`
: Build shared library for dynamic loading.

`INSTALL`
: Install add-on packages.

`REMOVE`
: Remove add-on packages.

`build`
: Build (that is, package) add-on packages.

`check`
: Check add-on packages.

`LINK`
: (UNIX only) Front-end for creating executable programs.

`Rprof`
: Post-process R profiling files.

`Rdconv`
: Convert Rd format to various other formats, including HTML, Nroff, LaTeX, plain text, and S documentation format.

`Rd2dvi`
: Convert Rd format to DVI/PDF.

`Rd2txt`
: Convert Rd format to text.

`Sd2Rd` Convert S documentation to Rd format.

`config` (UNIX only) Obtain configuration information.

Use

```
R CMD command --help
```

to obtain usage information for each of the tools accessible via the `R CMD` interface.

B.2 Invoking R under Windows

There are two ways to run R under Windows. Within a terminal window (e.g. `cmd.exe` or `command.com` or a more capable shell), the methods described in the previous section may be used, invoking by `R.exe` or more directly by `Rterm.exe`. (These are principally intended for batch use.) For interactive use, there is a console-based GUI (`Rgui.exe`).

The startup procedure under Windows is very similar to that under UNIX, but references to the 'home directory' need to be clarified, as this is not always defined on Windows. If the environment variable `R_USER` is defined, that gives the home directory. Next, if the environment variable `HOME` is defined, that gives the home directory. After those two user-controllable settings, R tries to find system defined home directories. It first tries to use the Windows `"personal"` directory (typically `C:\Documents and Settings\username\My Documents` in Windows XP). If that fails, and environment variables `HOMEDRIVE` and `HOMEPATH` are defined (and they normally are under Windows NT/2000/XP) these define the home directory. Failing all those, the home directory is taken to be the starting directory.

Environment variables can be supplied as '*name=value*' pairs at the end of the command line.

The following additional command-line options are available when invoking `RGui.exe`.

'`--mdi`'
'`--sdi`'
'`--no-mdi`'
: Control whether `Rgui` will operate as an MDI program (the default, with multiple child windows within one main window) or an SDI application (with multiple top-level windows for the console, graphics and pager).

'`--debug`' Enable the "Break to debugger" menu item in `Rgui`, and trigger a break to the debugger during command line processing.

In Windows with `R CMD` you may also specify your own '`*.bat`' or '`*.exe`' file instead of one of the built-in commands. It will be run with the following environment variables set appropriately: `R_HOME`, `R_VERSION`, `R_CMD`, `R_OSTYPE`, `PATH`, `PERL5LIB`, and `TEXINPUTS`. For example, if you already have '`latex.exe`' on your path, then

```
R CMD latex.exe mydoc
```

will run LaTeX on '`mydoc.tex`', with the path to R's '`share/texmf`' macros added to `TEXINPUTS`.

B.3 Invoking R under Mac OS X

There are two ways to run R under Mac OS X. Within a Terminal.app window by invoking `R`, the methods described in the previous sections apply. There is also console-based GUI (`R.app`) that by default is installed in the `Applications` folder on your system. It is a standard double-clickable Mac OS X application.

The startup procedure under Mac OS X is very similar to that under UNIX. The 'home directory' is the one inside the R.framework, but the startup and current working directory are set as the user's home directory unless a different startup directory is given in the Preferences window accessible from within the GUI.

Appendix C The command-line editor

C.1 Preliminaries

When the GNU **readline** library is available at the time R is configured for compilation under UNIX, an inbuilt command line editor allowing recall, editing and re-submission of prior commands is used. Note: this appendix does **not** apply to the GNOME interface under UNIX, only to the standard command-line interface.

It can be disabled (useful for usage with ESS[1]) using the startup option '`--no-readline`'.

Windows versions of R have somewhat simpler command-line editing: see '`Console`' under the '`Help`' menu of the GUI, and the file '`README.Rterm`' for command-line editing under `Rterm.exe`.

When using R with **readline** capabilities, the functions described below are available.

Many of these use either Control or Meta characters. Control characters, such as *`Control-m`*, are obtained by holding the ⟨CTRL⟩ down while you press the ⟨m⟩ key, and are written as *`C-m`* below. Meta characters, such as *`Meta-b`*, are typed by holding down ⟨META⟩[2] and pressing ⟨b⟩, and written as *`M-b`* in the following. If your terminal does not have a ⟨META⟩ key, you can still type Meta characters using two-character sequences starting with *`ESC`*. Thus, to enter *`M-b`*, you could type ⟨ESC⟩⟨b⟩. The *`ESC`* character sequences are also allowed on terminals with real Meta keys. Note that case is significant for Meta characters.

C.2 Editing actions

The R program keeps a history of the command lines you type, including the erroneous lines, and commands in your history may be recalled, changed if necessary, and re-submitted as new commands. In Emacs-style command-line editing any straight typing you do while in this editing phase causes the characters to be inserted in the command you are editing, displacing any characters to the right of the cursor. In *vi* mode character insertion mode is started by *`M-i`* or *`M-a`*, characters are typed and insertion mode is finished by typing a further ⟨ESC⟩.

Pressing the ⟨RET⟩ command at any time causes the command to be re-submitted.

Other editing actions are summarized in the following table.

C.3 Command-line editor summary

Command recall and vertical motion

`C-p` Go to the previous command (backwards in the history).

`C-n` Go to the next command (forwards in the history).

`C-r text` Find the last command with the *text* string in it.

On most terminals, you can also use the up and down arrow keys instead of *`C-p`* and *`C-n`*, respectively.

Horizontal motion of the cursor

`C-a` Go to the beginning of the command.

`C-e` Go to the end of the line.

`M-b` Go back one word.

[1] The 'Emacs Speaks Statistics' package; see the URL http://ESS.R-project.org

[2] On a PC keyboard this is usually the Alt key, occasionally the 'Windows' key.

`M-f`	Go forward one word.
`C-b`	Go back one character.
`C-f`	Go forward one character.

On most terminals, you can also use the left and right arrow keys instead of `C-b` and `C-f`, respectively.

Editing and re-submission

`text`	Insert *text* at the cursor.
`C-f` *`text`*	Append *text* after the cursor.
⟨DEL⟩	Delete the previous character (left of the cursor).
`C-d`	Delete the character under the cursor.
`M-d`	Delete the rest of the word under the cursor, and "save" it.
`C-k`	Delete from cursor to end of command, and "save" it.
`C-y`	Insert (yank) the last "saved" text here.
`C-t`	Transpose the character under the cursor with the next.
`M-l`	Change the rest of the word to lower case.
`M-c`	Change the rest of the word to upper case.
⟨RET⟩	Re-submit the command to R.

The final ⟨RET⟩ terminates the command line editing sequence.