The R System – An Introduction and Overview

J H Maindonald

Centre for Mathematics and Its Applications
Australian National University. http://www.maths.anu.edu.au/~johnm

Current draft: October 28, 2010

Languages shape the way we think, and determine what we can think about.
[Benjamin Whorf.]

S has forever altered the way people analyze, visualize, and manipulate data... S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers.
[From the citation for the 1998 Association for Computing Machinery Software award.]

Contents

Introduction

Note the following web sites:

CRAN (Comprehensive R Archive Network): `http://cran.r-project.org`

To obtain R and associated packages, use the nearest mirror.
`http://mirror.aarnet.edu.au/pub/CRAN` or `http://cran.ms.unimelb.edu.au/`.

R homepage: `http://www.r-project.org/`

Wikipedia: `http://en.wikipedia.org/wiki/R_(programming_language)`

R-downunder: `http://www.stat.auckland.ac.nz/mailman/listinfo/r-downunder`

For other useful web pages, click on the menu item R help, and look under Resources on the browser window that pops up.

Commentary on R

General

R runs on many types of system – Windows, Mac, Unix and Linux. It is free. Obtain it from a CRAN site (see above). It has extensive graphical abilities that are tightly linked with its analytic abilities. Much of the power of R for statistical analysis and for specialist graphics comes from the extensive enhancements that the packages build on top of the base system.

Other points are:

Although now relatively mature, the system gets continuing scrutiny, with improvements and enhancements appearing with each new release, i.e., every few months.

Though not perfect in this respect (!), the system has been developed with a keen regard to notions of good statistical practice.

Users should expect to encounter demands to improve their statistical knowledge, in order to use R effectively. The R community expects users to be serious about data analysis, to want more than a quick cook-book fix!

Statistical and allied professionals who wish to develop or require access to cutting edge tools find R especially attractive. It is also finding wide use among working scientists who have substantial and continuing data analysis problems that justify time spent in the mastery of R.

The base system and the recommended packages get unusually careful scrutiny. Nevertheless, there are traps. Take particular care with newer abilities, which may not have been much tested in regular use. Some of the contributed packages may not have been much tested, unless by their developers. The greatest risks arise from inadequate understanding of the statistical issues. [Such warnings apply, of course to any statistical system.]

Getting help

Although there is no official support for R, the r-help mailing list serves as an informal support network that can be highly effective. Details of this and other lists are on the home page for the R project: `http://www.r-project.org`. Note also the R-downunder list; for details go to `http://www.stat.auckland.ac.nz/mailman/listinfo/r-downunder`. Be sure to check the available documentation before posting to r-help. Archives are available that can be searched for questions that have been previously answered.

Use of an editor as a run-time environment

The Windows implementation, and the Cocoa based GUI for Mac OS X, now offer a simple script editor that has a Run Line or Selection feature. There are various editors and associated interfaces to R that allow editing of code, again offering a single click Run Line or Selection. On Windows systems, the Tinn-R editor (`http://www.sciviews.org/Tinn-R/`) is an excellent option. ESS (Emacs Speaks Statistics), now fully operational for Windows as well as for Unix, is attractive for users who relish the power of the Emacs editor.

The development model, and development strategies

The R system uses an open source development model that is broadly similar to that of Linux.[1] Its developer skill base is impressive.

A large effort has gone into the providing interfaces into other systems – Python, SQL and other databases, parallel computing using MPI, and Excel using the DCOM software.

Unifying ideas

Generic functions for common tasks – print, summary, plot, etc. (the Object-oriented idea; do what that "class" of object requires)

Formulae, for specifying graphs, models and tables.

Expressions can be:

evaluated (of course)

printed on a graph (come to think of it, why not?)

Language structures can be manipulated, just like any other object (Manipulate formulae, expressions, argument lists for functions, . . .)

Trellis (lattice) graphics – graphs whose layout reflects data structure

There are many unifying computational features, e.g.

Any 'linear' model (lm, lme, etc) can use spline basis functions to fit spline terms. This extends to any other system of basis functions.

These ideas are not uniformly implemented right through R, reflecting the incremental manner in which R has developed.

Retrospect, prospect and alternatives to R

Ross Ihaka and Robert Gentleman, both at that time from the University of Auckland, developed the initial version of R, for use in teaching tool. It implements a dialect of the S language that was developed at AT&T Bell Laboratories for use as a general purpose scientific language, but with especial strengths in data manipulation, graphical presentation and statistical analysis. Since mid-1997, development has been overseen by a 'core team' of about a dozen people, drawn from many different institutions worldwide.

The commercial S-PLUS implementation of S popularized the S language, giving it a large user base among statistical professionals and skilled scientific users. The existence of a large user base into which R could tap was helpful in getting a critical mass of R users in the early stages of its

[1] Observe that, whereas Linux competes in the shadow of Microsoft, R is not obviously in the shadow of any other system!

development. Its continuing success has come a development model that has fostered cooperative effort between statistical computing experts from many different parts of the world.

Other roughly comparable systems that might potentially have been the basis for an R-like project include the commercial Matlab system, Scilab, Octave, Gauss, Python and Lisp-Stat. Note the popularity of Matlab in the signal and image processing community.

Although with a syntax that looks superficially like that of C, the implementation of R has been heavily influenced by LISP. The R interpreter uses a model that is based on the Scheme dialect of LISP. Luke Tierney, and several others who had previously had a heavy involvement with Luke Tierney's Lisp-Stat system, are now actively involved in the ongoing development of R. See Tierney (2005), and other papers in the same volume of the *Journal of Statistical Software*

With the release of version 1.0 in early 2000, R became a serious tool for professional use. Since that time, the pace of development has been frenetic, with a new package appearing every week or two. There are now more than 800 packages available through the CRAN (Comprehensive R Archive Network) sites. Books that were specifically devoted to R began to appear in 2002.

Novice users will notice small but occasionally important differences between R and S-PLUS. Writers of substantial functions and (especially) packages will find larger differences. R's packages are now more wide-ranging in scope than S-PLUS libraries. Some specialised S-PLUS abilities may not be available in R or in R packages.

The R system uses a language model that dates from the 1980s. The body of code that has been build on top of base R is now so large that any change to a more modern language model will be difficult. Progress is likely to be evolutionary, building on and extending present abilities and high level R language constructs. Details of the underlying computer implementation will inevitably change, perhaps at some point radically.

Data set size, and databases

R's evolving technical design has allowed it, taking advantage of advances in computing hardware, to steadily improve its handling of large data sets. An important step was the move, with the release of version 1.2, to a dynamic memory model. The flexibility of R's memory model does however have a cost for some computations, relative to systems that are highly efficient in the processing of data from file to file. The difference in cost may however be small or non-existent for systems that have a 64-bit address space.

The R system has packages that provide links into a variety of types databases. An SQLite database can be created from within R, as shown in Subsection 6.6.3.

The statistics of data collection

The scientific context, which includes available statistical methodology, has crucial implications for the experiments that it is useful to do, and for the analyses that are meaningful. There are, in addition, constraints and opportunities that arise from computing software and hardware.

Statistics of data collection encompasses statistical *experimental design*, sampling design, and more besides. At base, the same issues arise in field, industrial, medical, biological and laboratory experimentation. The aim, as always, is to get maximum value from the use of all resources. The planning that is required will be most effective if based on sound knowledge of the materials and procedures used by experimenters. As we learn more about these issues, we gain the knowledge needed to design better experiments.

Documentation

Official Documentation: Users who are working through these notes on their own should have available for reference the document
"An Introduction to R", written by the R Development Core Team. To download an up-to-date copy, go to CRAN.

Web-based Documentation: See Documentation on the web page `http://www.r-project.org`

Note the R Wiki (`http://wiki.r-project.org/rwiki/doku.php`) and the extensive collection of help information that is listed under Other (`http://www.r-project.org/other-docs.html`).

For examples of R graphs, see `http://addictedtor.free.fr/graphiques/`.

R News: Successive issues of *R News* contain much useful information. These can be copied down from one of the CRAN sites.

Contributed Documentation: There is an extensive collection of user-written documents on R that can be accessed by going to this same mirror site, and clicking (under Documentation) on **Contributed**. See also the links that John Fox gives on the web page for his book that is noted under the reference for his book.

Books: Appendix B includes references to a number of books. Recently, a number of new books on R have appeared. See `http://www.R-project.org/doc/bib/R.bib` for a list that is updated regularly.

There will be occasional reference to

DAAGUR: Maindonald, J. H. & Braun, J. B. 2007. Data Analysis & Graphics Using R. An Example-Based Approach. Cambridge University Press, Cambridge, UK, 2007.
`http://www.maths.anu.edu.au/~johnm/r-book.html`

Chapter 1

Preliminaries

1.1 Installation of R and of R Packages

Installation of R First download and install R from a CRAN site. In Australia, go to:

`http://cran.ms.unimelb.edu.au/`

Windows and MacOS X users should download the relevant executable,
(e.g. **R-2.12.0-win32.exe** for Windows, or **R-2.12.0.dmg** for MacOS X).
Click on the downloaded file to start installation

Installation of R Packages (Windows & MacOS X)

Start R (e.g., click on the R icon). Then use the relevant menu item
to install packages via an internet connection.
This is (usually) easier than downloading, then installing.

Command line instructions can alternatively be used to install packages. See below.

Locating packages The CRAN task views may be a good first place to go.

For installation, follow the instructions in the text box. For installing packages, users may need to specify a mirror site. In Australia, specify the Australian mirror.

A fresh install is typically required to take advantage of new major releases (e.g. moving from a 2.11 series release to a 2.12 series vrelease) when they appear. For working through these notes, version 2.11.0 or later should be installed.

Installation of packages from the command line

For packages where there are dependencies, installation from the command line may be an attractive way to go. First, start R, perhaps by clicking on an R icon. Make sure that you have a live internet connection.

To install the R Commander from the comamnd line, enter:

```
install.packages("Rcmdr", dependencies=TRUE)
```

Among the dependencies are the graphics packages *rgl* (3D dynamic graphics), *scatterplot3d*, *vcd* (visualization of categorical data) and *colorspace* (for generation of color palettes, etc).

1.2 The R Commander Graphical User Interface, and Alternatives

This section will describe use of the R Commander. The *rattle* GUI will be discussed briefly in Subsection 13.1. Subsection 13 also makes brief mention of *JGR* (Java Graphics for R) and *pmg* (Poor Man's GUI). Note also the abilities that the *playwith* and *latticist* provide for interaction with graphs. These are both discussed in Subsection 7.3.6. See also Figure 7.12

1.2.1 The R Commander GUI – A Guided to Getting Started

The R commander gives access to a wide range of abilities, in the base R system and in R packages. Novices may find it especially helpful for data entry, and for graphics. It has interfaces to key abilities from the *lattice* and *rgl* packages, as well as from base graphics.

To start the R commander, start R and enter:[1]

```
library(Rcmdr)
```

This opens an R Commander script window, with the output window underneath. This window can be closed by clicking on the × in the top left corner. If thus closed, enter `Commander()` to reopen it again later in the session.

From GUI to writing code: The R commander displays the code that it generates. Users can take this code, modify it, and re-run it. The code can be run either from the R Commander script window or from the R console window (if open).

The active data set: The R Commander has, at any one time, a single "active" data set. Start by clicking on the Data drop-down menu. Here are alternative ways to select or create or change the active data set:

- Click on Active data set, and pick from among data sets, if any, in the workspace.
- Click on Import data, and follow instructions, to read in data from a file. The data set is read into the workspace, at the same time becoming the active data set.
- Click on New data set ..., then entering data via a spreadsheet-like interface.
- Click on Data in packages, then on Read Data from Package, then select an attached package and choose a data set from among those included with the package.
- A further possibility is to load data from an R image (`.RData`) file; click on Load data set ...

Creating graphs: To draw graphs, click on the Graphs drop-down menu. Then

- Click on Scatterplot ... to obtain a scatterplot. This uses `scatterplot()` from the *car* package, which is an option rich interface to functions that are in base graphics.
- Click on X Y conditioning plot ... for lattice scatterplots and panels of scatterplots.
- Click on 3D graph to obtain a 3D scatterplot, using the R Commander function `scatter3d()` that is an interface to functions in the *rgl* package.

[1] At startup, the R Commander checks whether all the suggested packages, needed to use all its features, are available. If some are missing, then upon starting up, the R commander offers to install them. For installing such packages, there must be a live internet connection.

Statistics (& fitting models): Click on the Statistics drop down menu to get submenus that give summary statistics and/or carry out various statistical tests. This includes (under Contingency tables) tables of counts and (under Means) One-way ANOVA. Also, click here to get access to the Fit models submenu.

***Models:** Click here to extract information from model objects once they have been fitted. (NB: To fit a model, go to the Statistics drop down menu, and click on Fit models).

Chapter 2

An Overview of R

Command prompt (>)	Enter commands following the prompt, e.g. `> 2 + 2  # Calculate 2 + 2`
Quitting	To quit from R type `q()  # NB q(), not q`
Case matters	`volume` is different from `Volume`

Assignment	The assignment symbol is <-, e.g.
	`volume <- c(351, 955, 662, 1203, 557)`
	`# Store the column of numbers in volume`
	`# c = concatenate`

Help	The main help function is `help()`. Note `help(help)` and, e.g., `help(plot)`
Other topics	Simple arithmetic operations; simple plots; input of data from a file.

2.1 Use of the console (i.e., command line) window

The command line prompt, i.e. the >, is an invitation to start entering commands. For example, type `2+2` and press the Enter key. The following appears on the screen:

```
> 2+3
[1] 5
>
```

The result is 5. The `[1]` says, a little strangely, "first requested element will follow". Here, there is just one element. The > indicates that R is ready for another command.

Try also:

```
> result <- 2+5
> result              # Check the contents of 'result'
[1] 7
>
```

The value 7 is now stored in an object with the name `result`. Objects such as `result` that have been created by the user go into what is called the *workspace*,

Note that:

- The assignment symbol is <-
- Typing the name of an object causes the printing of its contents, as above when `result` was typed on the command line. This applies to functions as well as data objects. For example, try typing `q`, or `mean`.
- The # symbol indicates that what follows, on that line, is comment.
- Multiple commands may appear on a line, with the semicolon (;) as the separator.

The exit or quit command is

```
> q()
```

NB: Typing `q` on its own, without the parentheses, displays the text of the function on the screen.

A message will ask whether to save the workspace image. Clicking Yes (usually the safest option) will save the objects that remain in the workspace – any that were there at the start of the session (unless removed or overwritten) and any that have been added since. Assuming that the very short session above started with an empty workspace, the only object in the workspace will be `result`. The workspace is automatically reloaded when the R session is restarted.[1]

Commands may continue over more than one line. By default, the continuation prompt is

+

As with the > prompt, this is generated by R. Including it when code is entered will give an error!

For the names of R objects or commands, case is significant. Thus `Myr` (millions of years) differs from `myr`. (`Myr` is a column in the data frame `molclock`, used in Exercise 1 in Section 3.5).

On Windows systems, the Microsoft Windows conventions apply, and case does not distinguish file names. On Unix systems (the Mac OS X version of Unix is a partial exception) case in file names is significant.

Practice with R commands

Try the following

```
1:5          # The numbers 1, 2, 3, 4, 5
mean(1:5)
sum(1:5)     # Apply the sum function to the vector
             # of numbers 1, 2, 3, 4, 5
(1:5) > 2    # Returns FALSE FALSE  TRUE  TRUE  TRUE
             # Other relational operators are: >=, <, <=, ==, !=
(2:5)^10     # 2 to the power of 10, 3 to the power of 10, ...
log2(c(0.5, 1, 2, 4, 8)) # Values that differ by a factor of 2
                         # are, on this scale, one unit apart.
```

The R language has the standard types of abilities for evaluating arithmetic and logical expressions.

A wide variety of functions extends these basic arithmetic and logical abilities. Common functions include `print()`, `plot()` and `help()`. Type `help(plot)` to get help on the function `plot()`.

2.2 A Short R Session

We will work with the data set shown in Table 2.1:

[1] If more than one *working directory* has been created, any workspace that is reloaded will for the *working directory* for the new session.

	Volume (mm^3)	Weight (g)	type
Aird's Guide to Sydney	351.00	250.00	Guide
Moon's Australia handbook	955.00	840.00	Guide
Explore Australia Road Atlas	662.00	550.00	Roadmaps
Australian Motoring Guide	1203.00	1360.00	Roadmaps
Penguin Touring Atlas	557.00	640.00	Roadmaps
Canberra - The Guide	460.00	420.00	Guide

Table 2.1: Weights and volumes, for six Australian travel books.

Entry of columns of data from the command line

Data may be entered from the command line, thus:

```
volume <- c(351, 955, 662, 1203, 557, 460)
weight <- c(250, 840, 550, 1360, 640, 420)
```

Read the symbol `c` as "concatenate". It joins elements together into a vector, here numeric vectors.

Now store the descriptions in the character vector `description`:

```
description <- c("Aird's Guide to Sydney",
 "Moon's Australia handbook",
 "Explore Australia Road Atlas", "Australian Motoring Guide",
 "Penguin Touring Atlas", "Canberra - The Guide")
```

The end result is that objects `volume`, `weight` and `description` are stored in the workspace.

Listing the workspace contents

Use the function `ls()` to examine the current contents of the workspace:

```
> ls()
[1] "description" "volume"      "weight"
> ls(pattern="ume")   # All objects whose names include "ume"
[1] "volume"
> ls(pattern="^des")  # All objects whose names start with "des"
[1] "description"
```

Operations with `vectors`

Here are the values of `volume`

```
> volume
[1]  351  955  662 1203  557  460
```

Here are various arithmetic operations:

```
> # Final element of volume
> volume[6]
[1] 460
> ## Ratio of weight to volume, i.e., density
```

```
> round(weight/volume,2)
[1] 0.71 0.88 0.83 1.13 1.15 0.91
```

A simple plot

Figure 2.1 plots `weight` against `volume`, for the six Australian travel books. Note the use of the graphics formula `weight ~ volume` to specify the $x-$ and $y-$variables. It takes a similar from to the "formulae" that are used in specifying models, and in the functions `xtabs()` and `unstack()`.

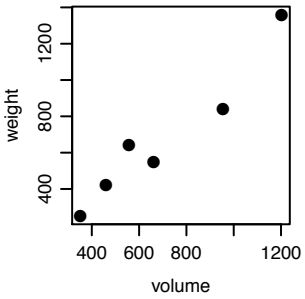

Figure 2.1: Weight versus volume, for six Australian travel books.

```
## Code
plot(weight ~ volume, pch=16, cex=1.5)
  # pch=16: use solid blob as plot symbol
  # cex=1.5: point size is 1.5 times default
## Alternative
plot(volume, weight, pch=16, cex=1.5)
```

The axes can be labeled:

```
plot(weight ~ volume, pch=16, cex=1.5, xlab="Volume (cubic mm)",
     ylab="Weight (g)")
```

Labeling of points (e.g., with species names) can be done interactively, using `identify()`:[2]

```
identify(weight ~ volume, labels=description)
```

Then click the left mouse button above or below a point, or on the left or right, depending on where you wish the label to appear. Repeat for as many points as required.

On most systems, the labeling can be terminated by clicking the right mouse button. On the Windows GUI, an alternative is to click on the word "Stop" (then on "Stop locator") that appears at the top left of the screen, just under "Rgui" on the left of the blue panel header of the R window.

Formatting and layout of plots

There are extensive abilities that may be used to control the formatting and layout of plots, and to add features such as special symbols, fitted lines and curves, annotation (including mathematical annotation), colors and so on. A later chapter (Chapter 7) is devoted to graphics.

[2] A non-interactive alternative is to use `text()` to place labels on all the points.

2.3 Data frames – Grouping together columns of data

Data Frames	
Data frames	Data frames are the preferred way to make data available to modeling functions.
Creating data frames	1: Enter from the command line, 2: Use `read.table()` to input from a file.
Accessing columns of data frames	`travelbooks$weight` or `travelbooks[, "weight"]` or `travelbooks[, 4]` Or: Use the `data` parameter, if available, in a function call Or: Use `with()`, e.g. `with(travelbooks, plot(weight  volume))` Use `attach()`, e.g., `attach(travelbooks)`, and `detach()` when finished.

Data frames are pervasive in R. Most datasets that are included with R packages are supplied as data frames. The following demonstrates the use of a data frame to group together, under the name `travelbooks`, the several columns of Table 1. It is tidier to have matched columns of data grouped together into a data frame, rather than separate objects in the workspace.

```
## Group columns together into a data frame
travelbooks <- data.frame(
   thickness = c(1.3, 3.9, 1.2, 2, 0.6, 1.5),
   width = c(11.3, 13.1, 20, 21.1, 25.8, 13.1),
   height = c(23.9, 18.7, 27.6, 28.5, 36, 23.4),
   weight = weight,  # Include values of weight, entered earlier
   volume = volume,  # Include values of volume, entered earlier
   type = c("Guide", "Guide", "Roadmaps", "Roadmaps", "Roadmaps",
            "Guide"),
   row.names = description
)
## Remove objects that are not now needed.
rm(volume, weight, description)
```

The vectors `volume`, `weight` and `description` had already been entered, and it was not necessary to re-enter them. It is a matter of convenience whether the description information is used to label the rows, or alternatively placed in a column of the data frame.

Vectors of character, such as `type`, are by default stored as factors. In the data as stored, `"Guide"` is replaced by 1 and `"Roadmaps"` by 2. Stored with the factor is the information that 1 is `"Guide"` and 2 is `"Roadmaps"`. In many contexts, factors are equivalent to character data. There are however situations where the difference is important.

Accessing the columns of data frames

The following all refer directly to the name of the data frame:

```
travelbooks[, 4]
travelbooks[, "weight"]
travelbooks$weight
travelbooks[["weight"]]   # This treats the data frame as a list.
```

However there are several mechanisms that avoid repeated reference to the name of the data frame. The following all plot `weight` against `volume`:

```
## 1: Use the data parameter in the function call
plot( weight ~ volume, data=travelbooks)
#
## 2: Use with(); take columns from the specified data frame
with(travelbooks, plot(weight ~ volume))
#
## 3: Use attach() to include the column names in the search list
attach(travelbooks)
plot( weight ~ volume)
detach(travelbooks)  # Detach when no longer required
```

Approaches 2 and 3 are always available. Most, but not all, plotting and modeling functions accept a `data` argument.

Subsection 3.3.2 will discuss the attaching of packages and image files.

2.4 Input of Data from a File

The function `read.table()` is designed for input from a file into a data frame. As an example, observe input of the data in Table 2.1. The *DAAG* package has a function `datafile()` that can be conveniently used to place into the working directory this and/or several other files that are intended for use for demonstrating input of data from a file.[3]

The first two lines (column headings and first row of data) are:

	thickness	width	height	weight	volume	type
Aird's Guide to Sydney	1.30	11.30	23.90	250	351	Guide
...						

Notice that the first column has no header information.

First store the file in the working directory, using the function noted above:

```
## Place the file in the working directory
library(DAAG)                    # DAAGxtras has the needed function
datafile("travelbooks")          # Place file in the working directory
dir()                            # List files in the working directory
file.show("travelbooks.txt")     # Display travelbooks.txt
```

Using `datafile()` to place the file in the working directory is purely a convenience for teaching purposes. Because the file is stored in the working directory the same command can be used, independent of the setup on individual computers, to read its contents into R.

For reading it into R, a suitable command is:

```
## Now input the file, to the data frame travelbooks
travelbooks <- read.table("travelbooks.txt", header=TRUE,
                          row.names=1)
  # Row 1 of the file gives column names. Column 1 gives row names
```

The assignment places the data frame in the workspace, with the name `travelbooks`. The row names are optional. The first seven columns are numeric. Because the final column holds character data, it is stored as a factor.

[3] *DAAG* must be installed.

Data input – points to note

Consider use or the R Commmander GUI. This displays entry boxes for input settings that users may find it expedient to change.

Use the parameter `heading` to control whether (`heading=TRUE`) or not (`heading=FALSE`) the first column of input is used for row names.

Section 6.6.1 comments on parameter settings that may need to be changed to match the data format. It also comments on what can go wrong, and makes suggestions on how to deal with various different types of input errors.

Character vectors that are included as columns in data frames become, by default, factors. For many purposes, character vectors and factors can be treated as equivalent.

2.5 Demonstrations, & Help Examples

Help in R

```
demo()           # List available demonstrations
demo(graphics)   # Demonstration of R's graphics abilities
##
example(plot)    # Run examples from help page for plot()
```

Note also `help.search()` and `apropos()`.

To get a list of available demonstrations, type:

```
demo()
```

Visually interesting demonstrations are:

```
demo(image)
demo(graphics)
demo(persp)
demo(plotmath)   # Mathematical symbols can be visually interesting
library(lattice)
demo(lattice)    # Demonstrates lattice graphics
library(vcd)     # The vcd package must of course be installed.
demo(mosaic)
```

Especially for `demo(lattice)`, it pays to stretch the graphics window to cover a substantial part of the screen. Place the cursor on the lower right corner of the graphics window, hold down the left mouse button, and pull.

The following gives a list of available demonstrations:

```
demo(package = .packages(all.available = TRUE))
```

Examples that are included on help pages

All functions have help pages. Most help pages include examples, which can be run using the function `example()`.

```
help()              # help on use of the help function
help(plot)          # the help page for the plot function
example(plot)       # Run the examples from the help page for plot()
par(ask=FALSE)      # Do not now ask, before displaying a new plot.
```

To work through the code for an example, look on the screen for the code that was used, and copy or type it following the command line prompt.

Be warned that, even for relatively simple functions, some of the examples may illustrate non-trivial technical detail.

Access to help resources from a browser screen

Type `help.start()` to display a screen that gives a browser interface to R's help resources. Note especially the listings under Frequently Asked Questions and Packages. Under Packages, click on base to get information on base R functions. Standard elementary statistics functions are likely to be found under stats, and base graphics functions under graphics.

Note the official R manuals. There is An Introduction to R, a manual on Writing R Extensions, and so on.

Many packages have vignettes; these are typically pdf files that give information on the package or on specific aspects of the package. Click on the package name, then on overview, to see links to any vignettes. Alternatively, note the function `vignette()`.[4]

Searching for key words or strings

Use `help.search()` to look for functions that include a specific word in their alias or title. For example, in order to look for a function for bar plots, try

```
help.search("bar")
```

This draws attention to the function `barplot()`. Type in `help(barplot)` to see the help page, and/or `example(barplot)` to run the examples.

Functions for operating on character strings are likely to have "str" or "char" in their name. Try

```
help.search("str", package="base")
help.search("char", package="base")
```

The function `RSiteSearch()` searches web-based resources, including R mailing lists, for the text that is given as argument.

2.6 Summary

One use of R is as a calculator, to evaluate arithmetic expressions. Calculations can be carried out in parallel, across all elements of a vector at once.

Use `q()` to quit from R. To retain objects created during the session, accept the offer to save the workspace.

Data frames collect together columns that all have the same length, as a single R object.

[4] Specify, e.g. `vignette(package="grid")` to get details of the vignettes that are available for the *grid* package. Then, to display the vignette, call `vignette()` with the package name (in character string form) as argument.

Attachment of a data frame (use the function `attach()`) can be convenient where a number of lines of code require access to its columns. Give the name without quotes.

The function `with()` attaches a data frame temporarily, for the duration of the call to `with()`. Where access to the dataframe columns is required for one or for a few lines only, this can be a good alternative to `attach()`.

For simple forms of scatterplot, use `plot()` and associated functions.

Useful help functions are `help()` (for getting information on a known function), `help.search()` (for searching for a word that is used in the header for the help file), and `apropos()` (for identifying functions that include a particular text string as part of their names). Note also the use of `help.start()`, to start a browser window from which R help information can be accessed.

`read.table()` is the function of first recourse for inputting rectangular files. As an alternative, consider use of the R Commander GUI.

2.7 Exercises

1. Use the function `datafile()` (*DAAG* or *DAAGxtras*), with the argument `file=bestTimes`, to place the file **bestTimes.txt** into the working directory.[5]

   (a) Examine the file. (Include the path if the file is not in the working directory.)

   ```
   file.show("bestTimes.txt")  # Assumes file in working directory
   bestTimes <- read.table("bestTimes.txt")
   ```

   (b) The `bestTimes` file has separate columns that show hours, minutes and seconds. Use the following to add the new column `Time`, then omitting the individual columns as redundant

   ```
   bestTimes$Time <- with(bestTimes, h*60 + min + sec/60)
     # Time in minutes
   names(bestTimes)[2:4]   # Check that these are the columns
                           # that can be omitted
   bestTimes <- bestTimes[, -(2:4)]  # Use "-" to omit these columns
   ```

   (c) Here are alternative ways to plot the data

   ```
   plot(Time ~ Distance, data=bestTimes)
   ## Now use a log scale
   plot(log(Time) ~ log(Distance), data=bestTimes)
   plot(Time ~ Distance, data=bestTimes, log="xy")
   ```

   (d) Now save the data into an image file in the working directory

   ```
   save(bestTimes, file="bestTimes.RData")
   ```

   For further explanation of the function `save()`, see the next chapter.

[5] Alternatively, copy it from the web page `http://www.maths.anu.edu.au/~johnm/datasets/text/` and place it in the working directory.

Chapter 3

The R Working Environment

The Working Environment of an R Session:	
Working directory	Files are by default read from this directory, or written to it
Workspace	This is the user's "database", where the user can make additions or changes, or delete objects, as desired.
Object	Objects include data, functions, formula objects, expression objects, ... Use `ls()` to list contents of current workspace.
Image files	Use to store R objects, e.g., workspace contents. (The expected file extension is **.RData** or **.rda**)
`save.image()`	Use to store or back up workspace congtents. Alternatively, use the relevant menu item. Make frequent saves!
Search list	Use `search()` to list the "databases" where R searches for objects. Use `library()` to add packages to the search list Use `attach()` to add a data frame or image file to the search list.

3.1 The Working Directory and the Workspace

The *working directory* is the directory where, in the current session, R by default looks for user files, and saves files that the user outputs. It pays to have a separate working directory, and associated workspace or workspaces, for each major project.

The workspace is, in R technical language, a "database" that holds all the objects that are under direct user control. The *workspace* holds, in the current session, objects that the user has created or input, or that were there at the start of the session and not later removed. The workspace is at the base of a list of "databases", known as the search list, that gives access to packages, objects in other directories, etc.

The workspace changes as objects are added or deleted or modified. It disappears at the end of the session, but a copy or "image" can and usually should be kept. Upon quitting from R (type `q()`, or use the relevant menu item), users are asked whether they wish to save the current workspace. The workspace is reloaded next time an R session is started in the same working directory.

Setting the Working Directory

When a session is started by clicking on a Windows icon, the icon's Properties specify the Start In directory. The default choice, usually an R installation directory, is not satisfactory for long-term use, and should be changed.[1]

It is good practice to use a separate working directory for each different project. On Windows systems, copy an existing R icon, rename it as desired, and change the Start In directory to the new working directory.[2]

3.2 Saving and retrieving R objects

Cautious users will from time to time save (back up) the current workspace image. The command `save.image()`) saves everything in the workspace, by default into a file named **.RData** in the working directory. Or, depending on the implementation, click on the relevant menu item.

Before making major changes in the workspace, it may be sensible to archive the contents of the current workspace, e.g., into a file with the name **archive.RData**. Specify

```
save.image(file="archive.RData")
```

Before exiting a session and saving the workspace, consider use of `rm()` to remove objects that are no longer required. Saving the workspace image will then save everything that remains.

Use `save()` to save one or more named objects into an image file. The following demonstrate the explicit use of `save()` and `load()` commands:[3]

```
save(volume, weight, file="books.RData")
  # Can save many objects in the same file
load("books.RData")          # Recover the saved objects
```

Actually, the function `save.image()` uses `save()` to perform a major part of its task.

An alternative to `load("books.RData")` is `attach("books.RData")`. This makes the objects available, but in a *database* that is separate from the user's workspace.

Writing data frames to text files

Use the function `write.table()` to write a data frame to a text file. More generally, to save several objects (data frames or any other R object) in the one file, use `dump()` (to save in a text format) or `save.image()`, as noted above.

[1] When a Unix or Linux command starts a session, the default is to use the current directory.

[2] The working directory can be changed once a session has started, either from the menu (if available) or from the command line. If the intention is change to a new workspace, you may first want to save the existing workspace, then typing `rm(list=ls()` to remove its contents. Then, once the working directory has been changed, load the new workspace.

[3] Objects may alternatively be "dumped" in a more human-readable dump format. See Subsection 6.7.

3.3 Installations, packages and sessions

Packages & the Search List	
Packages	Packages are collections of R functions and/or data. Most users install R from a binary on CRAN. Recommended packages are then installed along with R. Install other packages, as required, prior to their use.
`library()`	Use `library()` to attach a package, e.g., `library(DAAG)` Once attached, a package is added to the search list, i.e., to the list of "databases" that R searches for functions and/or data.
`attach()`	Use `attach()` to attach data frames or image (**.RData**) files. The data frame or image file is added to the search list, usually in position 2, i.e., following the workspace (`.Globalenv`)

3.3.1 The architecture of an R installation – Packages

An R installation is structured as a library of packages.

- All installations should have the base packages (one of them is called *base*) , which provide the infrastructure for other packages.
- Binaries that are available from CRAN sites include, also, all the recommended packages.
- Other packages can be installed as required.

A number of packages are by default attached at the start of a session. Other packages can be attached (use `library()`) as required. To discover which packages have been attached, enter:

```
sessionInfo()
```

Installation of R packages

From a Windows or MacOS X GUI, it is usually easiest to use the menu to install packages. This calls the function `install.packages()`. Alternatively, this function can be invoked directly. See `help(install.packages)`

Note also `download.packages()` (this takes a list of package names and a destination directory, downloads the newest versions of the package sources and saves them in 'destdir'), as zip or (under MacOS X) **.tar.gz** files. The menu, or `install.packages()`, can then be used to install the packages from the local directory.

For command line installation of packages that are in a local directory, call `install.packages()` with `pkgs` giving the files (with path, if necessary), and with the argument `repos=NULL`. If for example the binary **DAAG_1.00.zip** has been downloaded to **D:\tmp**, it can be installed thus

```
install.packages(pkgs="D:/DAAG_1.00.zip", repos=NULL)
```

In the R command line, be sure to replace the usual Windows backslashes by forward slashes.

On Unix and Linux systems, the relevant gzipped tar files, once downloaded to a storage device, can be installed using the shell command:

R CMD INSTALL <package (.tar.gz file)>

Note also the function `update.packages()`. This identifies packages for which updates are available, in each case offering the user the option to proceed with the update.

Use `.path.package()` to get the path of a currently attached package (by default for all attached packages).

3.3.2 The search path: library() and attach()

The *search path* determinines where and in what order R looks for objects (functions or data), required in an R session, that cannot be found in the workspace.

At any time in a session, the R system has a search path (or list) that determines where it looks for objects. To get a snapshot of the search path, type:

```
> search()
 [1] ".GlobalEnv"        "package:MASS"      "tools:RGUI"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

Technically, these are called "databases". R looks first in database 1 (`".GlobalEnv"`, which is the user workspace), then (if the object has not been found) in database 2, and so on.

Attachment of R packages

Use `library()` to attach an R package. This extends the search list. The system can then look in the package database for objects that are not in the user workspace.

If at some point (often the end of the session) the workspace is saved, and objects that were added have not been explicitly removed, they will be saved as part of the workspace. If saved in the default **.RData** image file in the working directory, they will be automatically loaded when a new session is next started in that working directory.

Attachment of image files

As noted earlier, the function `attach()` can be used to simplify access to

- columns of a data frames or elements of a list object
- objects that are stored in an image file.

The data frame or list object is added to the search list. Thus, columns of a data frame can be referred to by name, without explicit reference to the data frame. Be careful however not to double up on names that are already in the workspace.

The following demonstrates the attaching of an R image file:

```
attach("books.RData")
```

The session then has access to objects in the file **books.RData**. The file becomes a further "database" on the search list, separate from the workspace. Note however that if the object is modified, the modified copy becomes part of the workspace.

In order to detach such a database, proceed thus:

```
detach("file:books.RData")
```

Alternatively type `search()`, note the number that gives the position of the database on the search list, and supply that number as an argument to `detach()`.

3.4 Summary

Each R session has a working directory. This is the directory where R will by default look for files or store files that are external to R.

User-created R objects are added to the workspace, which is at the base of a search list, i.e., a list of "databases" that R will search when it looks for objects.

- At the end of a session an image of the workspace will typically (respond "y" when asked) be saved into the working directory. Additionally, it is good practice to save the workspace from time to time during a session. Before making big changes to the workspace, a useful precaution is to save the existing workspace under a name (e.g., `aug27.RData` that is different from the the default `.RData`

  It is usually best to keep a separate workspace and associated working directory, for each major project.

  The search path determines the order of search for objects that are accessed from the command line, or that a function requires and are not in the functions environment.

  Note also the use of `attach()` to give access to objects in an image (**.RData**) file. Include the name of the file (optionally preceded by a path) in quotes.

  R has an extensive help system. Use it!

3.5 Exercises

1. Read the data that is stored in the file **molclock1.txt** into the data frame `molclock`.[4]. Use the function `save()` to save the data into an R image file. Delete the data frame `molclock`, and check that you can recover the data by loading the image file.

[4]With the package `DAAGxtras` attached, typing `datafile()` will store **molclock1.txt**, **molclock2.txt**, and also **travelbooks.txt**, in your working directory