0

How to start a meaningful relationship with your computer (Introduction to R)

'Part of the inhumanity of the computer is that, once it is competently programmed and working smoothly, it is completely honest'

Isaac Asimov (1920–1992), author of science and fiction

This chapter looks and feels different to the rest of the book. It is short, contains no ecology and simply aims to familiarise you with the language used by scientific programmers and the particular conventions of R. It is not exhaustive, so all further R skills will be presented as needed in later chapters, in their appropriate mathematical, statistical and ecological context. The essential questions of what R is, why I chose to burden you with it and what it feels like to use it are covered in Sections 0.1–0.3. In Sections 0.4–0.7 you will find out where to obtain R and some of its valuable accessories, how to set them up in your computer and where to find help when you need it. I also outline the typesetting conventions that I will use to explain R code in this book. The last three sections (0.8–0.10) explain the basics of R usage and how to import data from other software into **data frames**.

0.1. What is R?

R is an open-source software package developed by a core team of academics and continually augmented by a large list of contributors. It is a numerical environment with a particular bias towards statistical analysis and modelling. To some extent, it is what you make of it. It may be used interactively to interrogate a data set or as a programming language to construct simulations and automate complicated tasks. Despite being free to academic users, R compares favourably with other data-analysis and modelling software. For example, it can do considerably more than basic proprietary software such as SPSS or MiniTab and it competes well with very expensive software such as SAS, MATHEMATICA and MATLAB.

0.2. Why use R for this book?

It is generally better to teach scientific computing using real rather than pseudo-code. It is much better to understand the lofty concepts of programming through a particular language, any language. It is then easier to cross over to another if it is better suited to your purposes. There are several accomplished environments for data analysis and scientific programming but there are several reasons why the choice of R for this book is particularly sound.

The first is its overall suitability to the workflow of ecologists. In the field of quantitative software, packages have historically belonged to one of three camps:

❶ traditional programming languages like Pascal, Fortran and C with basic numerical libraries;

❷ mathematical software like MATHEMATICA and MAPLE with extensive libraries for symbolic analysis;

❸ statistical software such as S, R and SAS with extensive libraries for data analysis.

Any one of these would be suitable, but since the majority of quantitative ecologists spend most of their careers analysing data sets and running simulation models (or solving analytical models numerically), software from the third category seems to fit best.

Another important reason for choosing a software tool, particularly considering the time and effort required to become proficient in it, is longevity. R has a respectable pedigree (its foundations were laid in the software S that has existed since 1976) and it also has considerable potential. Currently, the momentum behind R shows no signs of abating and this augurs well for its future. This momentum guarantees the continuous supply of contributed packages to do almost any imaginable task, books at various levels of specialisation, online resources and text editors for programming (see Section 0.6). Many of these tools and textbooks are aimed at, or motivated by, ecological applications.

Finally, R is freely distributed under the Gnu public licence. There are great ideological reasons for supporting a piece of software developed by publicly-funded academics who then freely distribute their work, placing it at the service of the worldwide academic community. The fact that it is also open-source means that the number of good brains working to improve it is likely to exceed those employed by a private software company.

0.3. Computing with a scientific package like R

Most of the tasks that an ecologist would care to do on other specialised packages (e.g. geographical information systems, spreadsheets, databases etc.) can also be done in R, with one crucial difference: because it is a programming language, R is considerably more flexible and customisable. Using the built-in commands and the additional packages that can readily be downloaded from the CRAN website, you can write computer code for any imaginable task. This comes at a price to user-friendliness: as with any large tool-box, you need to know what the tools are for, how to use them and in what order. For larger tasks that need to be done several times over, you will need to bundle together several tools in a well-defined sequence. This is called **programming**. If, in your career so far, you have only dealt interactively with a computer (ask a question, get back an answer, then ask another question, possibly based on the previous answer, and so on) then you might find that you need to shift your way of thinking about computers somewhat. Specifying complex tasks for a computer to do is an unforgiving and frequently frustrating job. Not only do you first need to perform the task manually (at least once) to make sure you know what you want done, but you then need to explain it to the computer unambiguously, in a language that looks nothing like written speech. Once this is done, you will often spend long hours looking at the screen wondering why on earth your apparently perfect piece of code comes up with an incomprehensible error message. The problem may be a tiny typo, a missing bracket or a fundamental logical inconsistency. Invariably, you will have to swallow the humiliation that, whatever the mistake was, it was yours and not the computer's. Even when these errors (or **bugs**) have been detected and fixed, there is always the possibility that the computer flawlessly performs a task other than the one you want. For example, a computer program may obligingly allow biological populations to recover from extinction long after their size has become negative. So, the process of **debugging** requires you to be untrusting and critical towards your own creation. This is probably one of the best life-lessons that your computer can teach you.

Once you have adjusted your expectations of how long it takes to develop a piece of code, things can only get better. You may start to enjoy the hunt for bugs, the creative process of constructing a functioning tool out of nothing, the rewarding feeling of uncovering the secrets of your data. Crucially, you will get better as a scientific programmer. You may even savour the rare occasion when code works perfectly the very first time you run it.

0.4. Installing and interacting with R

Day-to-day work with R involves the **R base package**, additional R packages as required, a good text editor and a quick-reference document of your liking. I explain what each of these is and how to obtain them.

The R base package contains the functionality required by most users including the basic user interface, mathematical, graphical and programming functions and all essential statistical tests and models. It can be downloaded from the Comprehensive R Archive Network (or CRAN for short) at http://cran.r-project.org/. You need to select the appropriate version for your operating system (Linux, Windows or Mac). You then need to follow the link to the base package and download the current version (v2.9.2 at the time of writing this book). Once prompted by a dialogue box, ask for the executable to be run and follow the default options in the various prompts of the setup program. When the program installs, start it up (e.g. by clicking at the desktop shortcut), you should see a screen like the one in Figure 0.1.

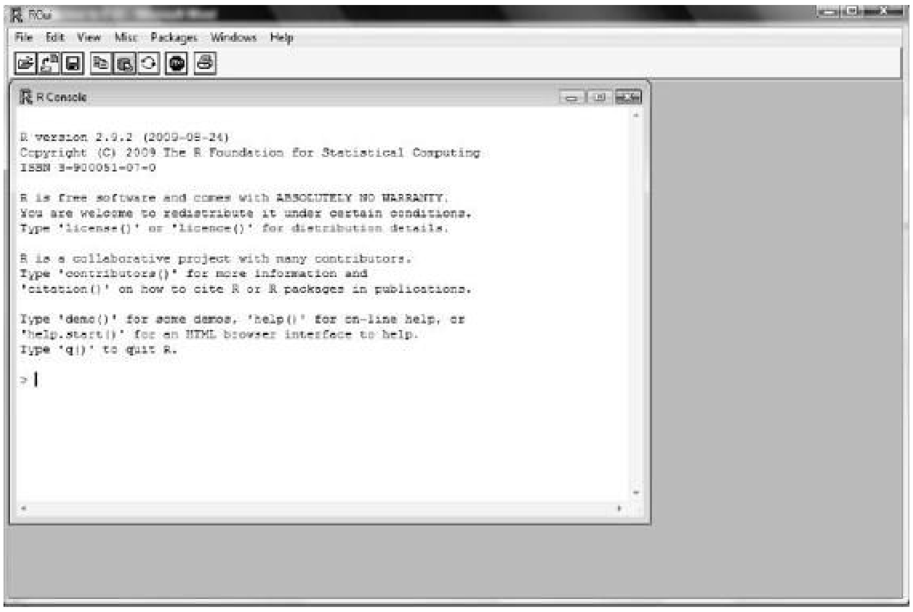
R uses a **command-line interface**, meaning that all the interesting stuff isn't done through the drop-down menu in the RGui window, but by typing commands next to the prompt (>) in the **R Console window**. When using it interactively, you type something at the prompt which can cause R to give you an output. Try typing something, say a numerical calculation, and then press return:

```
> 1+1
```

The response from R is

```
[1] 2
```

Figure 0.1: Start-up screen of the R command-line interface.



```
R version 2.9.2 (2009-08-24)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

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 and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

The serial number in square brackets indicates the order of a particular output line resulting from the previous user input. For example, to get R to print a list of all the years from my birth until writing this paragraph, the input and output would be:

```
> 1970:2009

 [1] 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980

[12] 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991

[23] 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002

[34] 2003 2004 2005 2006 2007 2008 2009
```

The colon (:) always indicates a range of values. Depending on the width of your screen, R will generate the list in as few lines as possible, quoting the serial number of the next item within square brackets, at the start of each line.

When your input is not understood by R, you will get an error message:

```
> this_input_is_rubbish

Error: object ‘this_input_is_rubbish’ not found
```

The flow of printed information in the R Console is always downwards. Although you can scroll up to view your previous workings, and you can use your mouse to highlight and copy bits of printed input/output, you cannot navigate up to edit any part of your previous inputs. You may, however, use the up arrow key on your keyboard to quickly copy a previously typed line of code onto the currently active prompt line. Working with such one-liners is not a problem within the R Console but it can get cumbersome when you need to input several lines of code together. In these cases, a better alternative is to type your code in a text editor (Section 0.6) and copy/paste it into the R Console.

Assignments in R can be done with the arrow symbols (`<-, ->`). For example, to give a name (say, `years`) to the above list of years, you would type

```
> years<-1970:2009
```

An assignment prompts no response from R. The information is simply stored under the name `years`, ready for later use. If you want to inspect the information, type `years` and press return.

R is **vectorised**, meaning that operations can be applied to entire collections of things with the same ease as applying them to single items. For example, my entire list of birthday anniversaries can be calculated (and filed under the name `ages`) as

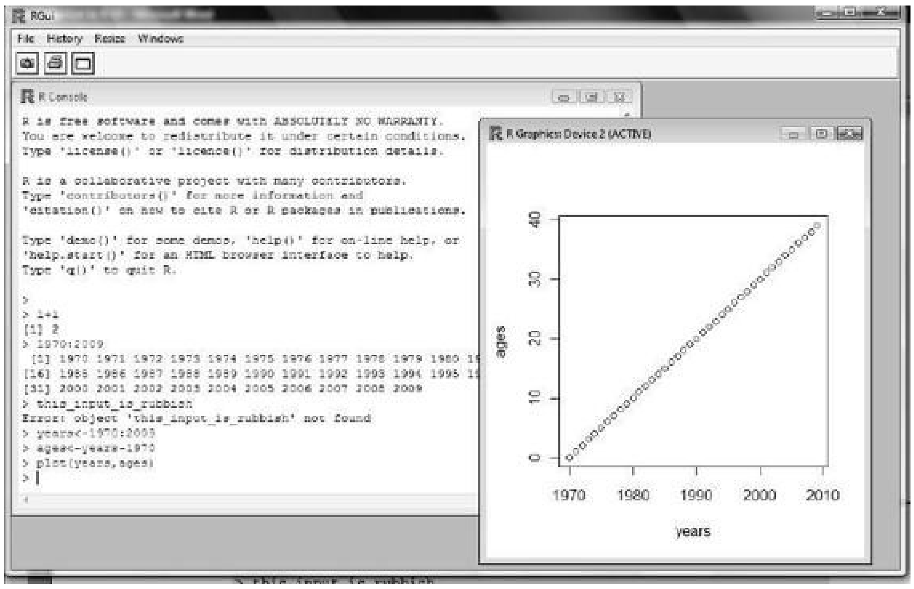
```
> ages<-years-1970
```

While the R Console handles the alphanumeric input and output, the **R Graphics Device** deals with images. This window appears separately within the RGui window. For example, a plot of ages versus years can be created by typing

```
> plot(years, ages)
```

Pressing return brings up the graphics window on screen (Figure 0.2). Feel free to rearrange and reposition this within your workspace.

Figure 0.2: R Gui, R Console and R Graphics Device: the three main components of the R interface.



```
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 and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
> 1+1
[1] 2
> 1970:2009
 [1] 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 19
[16] 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 19
[31] 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009
> this_input_is_rubbish
Error: object 'this_input_is_rubbish' not found
> years<-1970:2009
> ages<-years-1970
> plot(years,ages)
> |
```

0.5. Style conventions

When editing your code outside of R, it is a good idea to use monospaced fonts (such as `Courier`), because, unlike proportional fonts (like Times), they allocate equal space to all characters. This retains the spacing of some tabular forms of output and makes code easier to debug. In this chapter, I have placed R input and output on a grey background. In the rest of the book, I use a grey background for the entire R boxes to make the computing sections more obvious among the rest of the text. When describing interactive use of R, I precede user input by the prompt (>) and use boldface for the output:

```
> 1+1

[1] 2
```

Larger pieces of code, of a length that might be typed up in a text editor and then pasted into R, are presented without the prompts, accompanied by short comments in English for most lines. Such detailed annotation is good practice when programming and not just for the benefit of others: I am always surprised by how hard it is to understand my own uncommented code a mere few weeks after writing it. The special character # tells R to ignore the remainder of that line so that when copy/pasting code, the comments do not interfere with the computation. In the book, such comments are shown in italics. For example:

```
# This plots age v calendar year for a person born in 1970

years<-1970:2009   # A list of years since 1970

ages<-years-1970   # A list of ages since birth

plot(years,ages)    # Generates the plot
```

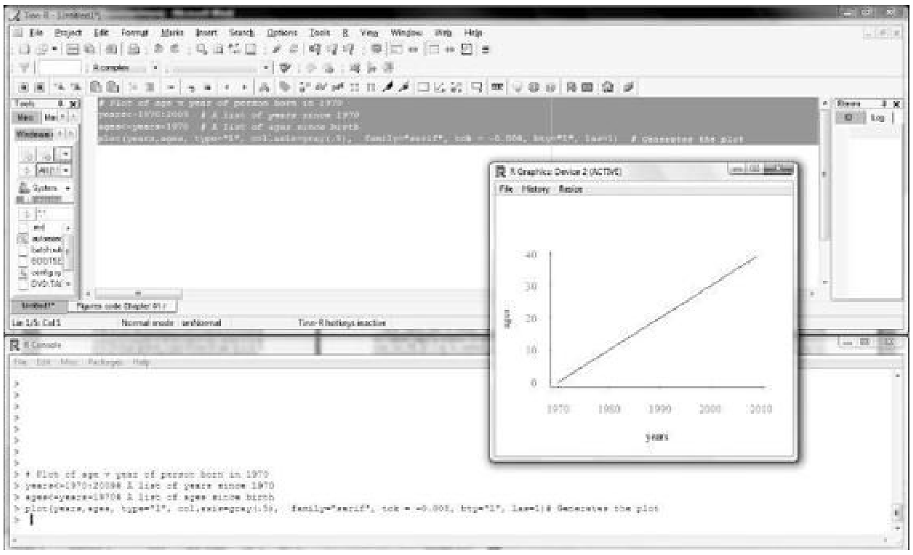
0.6. Valuable R accessories

The functionality of R can be expanded by installing additional packages. A **package** is a collection of additional functions, example data sets and documentation. Whenever this happens in this book, you will be informed which package to get, but you need to know how. There are two types of packages: those that only need to be loaded into R and those that require a full install (i.e. downloading from the CRAN site and then loading into R). Both can be done via the 'Packages' drop-down menu in the RGui window. Upon selecting 'Load package...' you will be presented with a selection of about 28 packages. Simply select the one you need and press OK. The package is then loaded and can be used by your current R session. Alternatively, packages not on this list can be obtained by selecting 'Install package(s)...', again from the 'Packages' drop-down menu. You may be asked to select a CRAN mirror site. Just pick the one that is closest geographically to you. This list of packages is considerably larger. Pick your package and wait until it downloads and expands. Installed packages are saved on the hard disc and stay with your computer even after your R session ends. You do, however, need to load them into R when you start a fresh session. Go through 'Packages->Load package...' as before. You will notice that your recently installed package has made its appearance in this list. An alternative to using the drop-down menu to load a package is to do it using the commands `require()` or `library()`. For example, `require(MASS)` will load the package MASS. If your code requires a package, then place the `require()` command right at the beginning, so that the package is loaded before the rest of the code is executed.

The constant introduction of new R commands throughout the book might leave you overwhelmed by the apparent arbitrariness of their names. Rest assured you are not alone. Each computer language may use different names for different purposes and no programmer can remember more than a small vocabulary. Navigating help files is therefore an essential skill (see Section 0.7) but equally important is a good quick-reference guide of the names and syntax of the most frequently used R commands. My personal favourite was created by Tom Short and can be found at http://cran.r-project.org/doc/contrib/Short-refcard.pdf.

Finally, there is the delicate issue of the text editor to be used for developing longer pieces of code. You do need one, but the choice is a matter of taste (Figure 0.3). Word-processors are to be avoided, because their spell-checking and slow searching facilities tend to get in the way. You may, instead, use a fast and simple text editor such as TextPad (freely available from http://www.textpad.com/products/textpad/index.html). Better still, you can download a text editor that has been developed specifically for R programming. More information on editors can be found at http://www.sciviews.org/_rgui/projects/Editors.html.

Figure 0.3: The combination of a specialist text editor with R can greatly facilitate programming work. Here, R is seen running together with Tinn-R, a great editor for the Windows operating system.



File Project Edit Format Marks Insert Search Options Tools R View Window Web Help
R Graphics: Device 2 (ACTIVE)
File History Resize
R Console
File Edit Misc Packages Help
ages
years

```
> # Plot of age v year of person born in 1970
> years<-1970:2009# A list of years since 1970
> ages<-years-1970# A list of ages since birth
> plot(years,ages, type="l", col.axis=gray(.5),  family="serif", tck = -0.005, bty="l", las=1)# Generates the plot
```

0.7. Getting help

There are three sources of information on various aspects of R. Printed and online manuals can inform you about the capabilities of the program and offer detailed, worked examples. Several pdf manuals come together with the R base package and you can access them from the ‘Help’ drop-down menu in the R Console. A list of further references can be found in Sections 0.13 and 0.14.

By working through books (such as the one you are holding), you will become aware of what is required of you as a programmer and what R can do for you as a scientist. The quick-reference guide mentioned in the previous section will keep you right regarding the syntax of commands for common tasks. However, you will regularly need to read more about the syntax and details of particular commands by researching the R help files. You can access these in two ways from within R. Go to the ‘Help’ drop-down menu and select ‘Html help’. This will launch your web browser to display a page for searching and browsing keywords. The contents of this page are stored on your hard disc, so don’t worry if you happen to be working off-line.

Alternatively, within the R Console, you can type a question-mark followed by the R command you want help on. For example, `?plot` will bring up, in a new window, the help file for the basic plotting command. You will initially find R help files somewhat...unhelpful. The way they present information takes some getting used to but, thankfully, surmising what you need becomes easier with practice. Each help file usually contains a section summarising the purpose of a command (‘`Description`’), its correct syntax (‘`Usage`’), its inputs (‘`Arguments`’) and outputs (‘`Value`’). Most importantly, towards the bottom, all help files have examples of usage (‘`Examples`’) and hyperlinks to relevant commands (‘`See also`’). If you are unsure of the name of the command for which you want help but you vaguely remember that it pertains to or contains some keyword, try typing `??keyword`. This performs a search through the help files for related commands. If the command cannot be found by the help searches, then it is possible that you are trying to read the help files for a package that you have not yet installed into R.

If all else fails you may seek information from the web. There are several searchable help archives on the CRAN website and Google will usually come up with the goods within its first page of search results. For questions on the base package, the forum **R-help** is a good starting point. The R community has a great record of responding to questions but before sending a question to the group, make sure that your question has not previously been answered in the archives.

0.8. Basic R usage

At its very simplest, R can be used as a glorified calculator. For example, the expression $\sqrt{(20-1.5)^2+(20-5)^2}$, can be calculated as

```
> sqrt((20-1.5)^2+(20-5)^2)

[1] 23.81701
```

Note the use of bracketing to specify which parts of the expression are to be squared and which are to be square-rooted. Brackets are used to specify the priority of operations. Hence, the expression

```
> (1+2)*4

[1] 12
```

does not give the same result as

```
> 1+2*4

[1] 9
```

Bracketing is also used for R commands. The following generates a pretty plot of the age v calendar year data

```
plot(years, ages, type="l", col.axis=gray(.5), family="serif",

tck=-0.008, bty="l", las=1)
```

Since this syntax is representative of most R commands, it is useful to elaborate: The name of the command (in this case, `plot`) comes first, followed by a pair of brackets containing the input to be used by the command and the options specifying how to use it. In this example, the inputs are two lists, for the x- and y-axis data (`years` and `ages`) followed by a total of six options. The name of the option comes first and its assigned value follows the equality sign (=). For example, `type = "l"` specifies the type of plot as a line plot (rather than a scatter plot), `col.axis = gray(.5)` specifies that the text used for the tick mark labels will be a medium shade of grey and `las = 1` that these labels will be horizontal (rather than parallel to the two axes). Two things must be noted: first, option assignments are not done by arrow signs (`<-, ->`), if you do, R will respond with an error message. Second, specifying options is *optional* because all options are set by the R developers to some default value. For example, the command `plot()` has about 70 possible options (type `?plot` and follow the link to `par` to see the complete list of graphical parameters) . You can use as many of them as you like to specify a plot to your exact standards, but omitting all the options and simply typing `plot(years, ages)` will generate a perfectly decent plot.

Other types of brackets are also employed by R. As we will see in later chapters, square brackets [ ] are used to modify and extract elements from data sets and curly brackets { } are used in programming to package together multiple lines of commands.

0.9. Importing data from a spreadsheet

Ecologists spend long days painstakingly collecting data in the field and long nights analysing them in front of a computer. It would be a shame if the analysis was spoiled simply because measurements of animal body weight were accidentally imported into the column for vegetation cover. One way to ensure this doesn't happen is to standardise the protocol for importing data.

The main R command for data import is `read.table()`. This requires information on the location of the file that holds the data and its specific format. Consider an Excel spreadsheet containing a column of years and a column of ages, as shown in Table 0.1.

Table 0.1

Years	*Age*
1970	0
1971	1
1972	2
1973	3
1974	4
1975	5
1976	6
1977	7
1978	8
1979	9
1980	10
1981	11

Because Excel spreadsheets have multiple sheets, it is easier to export a single sheet by saving it as a text file. In Excel, go to 'File→Save as...' and save the current sheet as 'Text (Tab-delimited)'. Assuming that the full path to the file is C:\My documents\Data\YearsAge.txt, then it can be imported into R by typing:

```
read.table("C:/My Documents/Data/YearsAge.txt", header=TRUE)
```

If the option `header` is set to `TRUE`, the first row of the spreadsheet is interpreted as a header, meaning, in this example, that R will recognise the first column by the name `Years` and the second by the name `Ages`. Note that the file path needs to be enclosed in double quotes and specified in terms of double backslashes. If you are likely to be importing different files every time, you may want to consider the following version of the command which launches a browsing dialogue box:

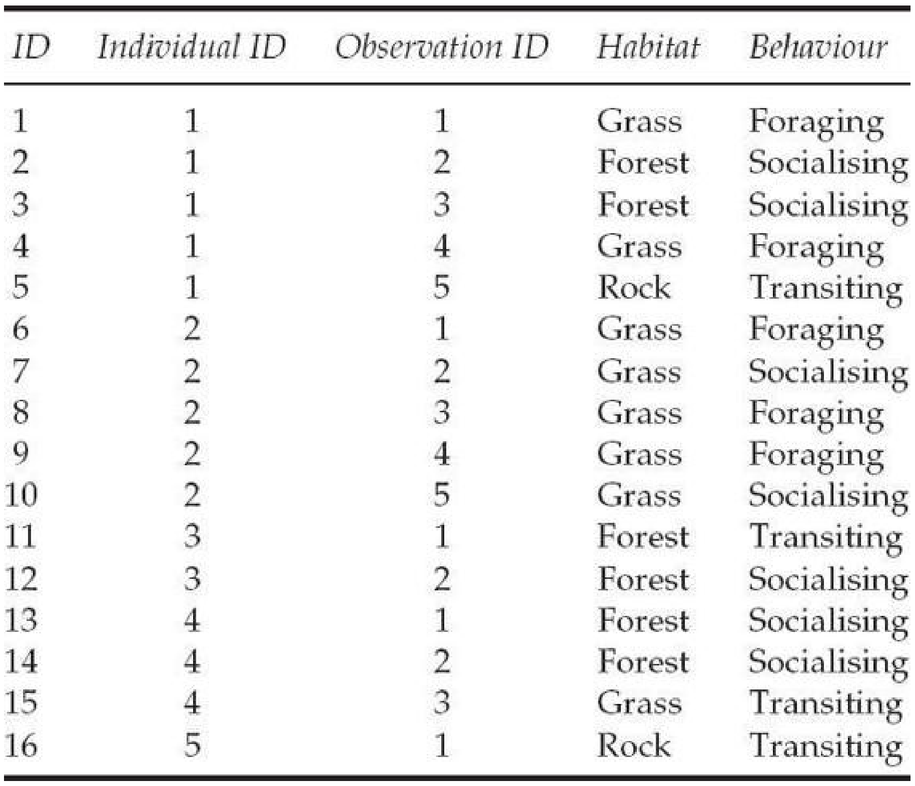
```
read.table(file.choose(), header=TRUE)
```

If you already have a nonstandard text file (one that is not tab-delimited) you can adapt the `read.table()` command so that R can read it. Options available for importing data can be found by typing `?read.table`. If you are still having problems, refer to the R Console manual 'R Data Import/Export'. You will find this through the '`Help`' drop-down menu, under the option '`Manuals (in PDF)`'.

0.10. Storing data in data frames

A **data frame** is a two-dimensional tabular object used for storing different types of data. The columns of a data frame store qualitatively different types of measurements and its rows correspond to sampling units or replicates. The data frame shown in Table 0.2 contains 16 observations from five animals sighted in different habitats, performing different behaviours. Although for the purposes of statistical analysis, it is debatable whether the appropriate sampling unit is the observation or the individual, the data frame must contain the full information of the data. Hence, the definition of replicate used for the rows of the data frame must represent the data in its most resolved form.

Table 0.2



ID	*Individual ID*	*Observation ID*	*Habitat*	*Behaviour*
1	1	1	Grass	Foraging
2	1	2	Forest	Socialising
3	1	3	Forest	Socialising
4	1	4	Grass	Foraging
5	1	5	Rock	Transiting
6	2	1	Grass	Foraging
7	2	2	Grass	Socialising
8	2	3	Grass	Foraging
9	2	4	Grass	Foraging
10	2	5	Grass	Socialising
11	3	1	Forest	Transiting
12	3	2	Forest	Socialising
13	4	1	Forest	Socialising
14	4	2	Forest	Socialising
15	4	3	Grass	Transiting
16	5	1	Rock	Transiting

The command `read.table()` automatically imports data into a data frame, so all you need to do is name it. For example, if the above data set is saved by Excel as a tab-delimited text file ('`Sights.txt`'), the data frame is created at import

```
data<-read.table("C:/My Documents/Data/Sights.txt", header=TRUE)
```

It is good to check that the data sheet imported has the right number of rows and columns

```
> nrow(data)

[1] 16
```

```
> ncol(data)

[1] 5
```

R has a wealth of commands for manipulating data frames. To see the column names of the data frame type

```
> names(data)

[1] "ID" "Individual" "Indiv_ID" "Habitat" "Behaviour"
```

To extract any one of your columns, you may call it by using the following syntax

```
> data$Habitat

 [1] Grass Forest Forest Grass Rock Grass Grass Grass Grass

[10] Grass Forest Forest Forest Forest Grass Rock

Levels: Forest Grass Rock
```

Here, R has identified that the variable Habitat takes a discrete number of values. Statisticians call such variables **factors** and R has taken it upon itself to identify the three values (or **levels**) that this variable has taken. An even easier way to access the content of the data frame is to attach it to the R search path. You only need to do this once in any one session and it allows data frame columns to be called directly by name.

```
> attach(data)

> Habitat

 [1] Grass Forest Forest Grass Rock Grass Grass Grass Grass

[10] Grass Forest Forest Forest Forest Grass Rock

Levels: Forest Grass Rock
```

This leaves open the possibility for naming conflicts: e.g. there may be more than one attached data frame with a column called `Habitat`. R will give you warnings if one attached data frame is about to mask the contents of another. To avoid these issues, you can `detach()` a data frame when you are finished with it.

Specific segments of the data frame can be extracted by specifying row and column numbers (in that order) inside square brackets. I will talk a lot more about these conventions in Chapters 1 and 6 but here are a few examples. A particular row (say the fifth), can be extracted as follows:

```
> data[5,]

    ID Individual Indiv_ID Habitat   Behaviour

5   5          1         5     Rock Transiting
```

Here, the column reference is left vacant after the comma in the square brackets, meaning that all column values for that row are required. A range of rows (say, all observations from the first animal) can also be obtained using the colon notation:

```
> data[1:5,]

    ID Individual Indiv_ID  Habitat    Behaviour

1   1           1         1   Grass      Foraging

2   2           1         2  Forest   Socialising

3   3           1         3  Forest   Socialising

4   4           1         4   Grass      Foraging

5   5           1         5    Rock   Transiting
```

The data frame can also be queried using conditioning. For example, all the rows that refer to the second animal can be found as follows:

```
> data[Individual==2,]

     ID Individual Indiv_ID  Habitat  Behaviour

6     6          2        1  Grass    Foraging

7     7          2        2  Grass Socialising

8     8          2        3  Grass    Foraging

9     9          2        4  Grass    Foraging

10   10          2        5  Grass Socialising
```

0.11. Exporting data from R

Some analyses take a long time to design. Others take a long time to run. Their numerical results may therefore be valuable enough to store on the hard disc via the command `write.table()`. At its simplest, you are required to specify the data frame to be saved to file and the full path. For example, the line,

```
> write.table(data, file="C:/My Documents/Data/SightsE.txt",

col.names=TRUE)
```

will write the data into the file `SightsE.txt`. If you would like to browse for the target location of your new file type

```
> write.table(data, file.choose(), col.names=TRUE)
```

During a new session the data can be re-acquired by typing

```
> data<-read.table("C:/My Documents/Data/SightsE.txt",

header=TRUE)
```

Communicating with other users, not all of whom may use R, will require you to export your data in formats readable by other packages. You can do this by customising the options in `write.table()`. For example, here is how to create an Excel file called `SightsE.xls`

```
> write.table(data, "C:/My Documents/Data/SightsE.xls",

sep="\t", row.names=FALSE, col.names=TRUE)
```

The special character \t uses tab as a column separator and the appropriate file extension (here, `.xls`) ensures that, once double-clicked, the file will be opened by Excel.

0.12. Quitting R

You can end your R session either by closing the R Console, or by typing `q()`. You will be asked if you want to save your workspace image. This will store all the imported data and variable names so that you find yourself exactly where you were when you next open R. This sounds appealing but it is best avoided because R then tends to accumulate a lot of definitions from previous analyses that take up memory and may lead to conflicts. If you have saved your previous sessions and want to wipe the slate clean you can do it by typing

```
rm(list=ls())
```

Further reading

The core developers of R have, over the years, produced several introductory references. Prime examples are Venables and Smith (2002) and Dalgaard (2008). Other, general references are Crawley (2005, 2007) and Zuur *et al.* (2009). Recently, several example-based books have appeared (Everitt and Hothorn, 2006; Braun and Murdoch, 2007) that are great for people who like to adapt worked solutions to their own needs. If you are looking for R books with an ecological bias, check out Bolker (2008) and Stevens (2009).

References

Bolker, B. (2008) *Ecological Models and Data in R*. Princeton University Press. 408pp.

Braun, W.J. and Murdoch, D.J. (2007) *A First Course in Statistical Programming with R*. Cambridge University Press, Cambridge. 163pp.

Crawley, M.J. (2005) *Statistics: An Introduction using R*. John Wiley & Sons, Ltd, Chichester. 342pp.

Crawley, M.J. (2007) *The R Book*. John Wiley & Sons, Ltd, Chichester. 942pp.

Dalgaard, P. (2008) *Introductory statistics with R*. Springer. 364pp.

Everitt, B.S. and Hothorn, T. (2006) *A Handbook of Statistical Analyses using R*. Chapman & Hall/CRC, Boca Raton. 275pp.

Stevens, M.H.H. (2009) *A Primer of Ecology with R*. Springer. 388pp.

Venables, W.N. and Smith, D.M. (2002) *An Introduction to R*. Network Theory Ltd. 156pp.

Zuur, A.F., Ieno, E.N. and Meesters, E.H.W.G. (2009) *A Beginner's Guide to R*. Springer. 220pp.