How to be a Quantitative Ecologist

The 'A to R' of Green Mathematics and Statistics

Jason Matthiopoulos

WILEY

How to be a Quantitative Ecologist

How to be a Quantitative Ecologist

The 'A to R' of Green Mathematics and Statistics

Jason Matthiopoulos
University of St Andrews, Scotland, UK

WILEY
A John Wiley & Sons, Ltd., Publication

This edition first published 2011

Library of Congress Cataloging-in-Publication Data

Matthiopoulos, Jason.
How to be a quantitative ecologist : the 'A to R' of green mathematics and statistics / Jason Matthiopoulos.
p. cm.
Summary: ''The book will comprise two equal parts on mathematics and statistics with emphasis on quantitative skills'' – Provided by publisher.
Includes bibliographical references and index.
ISBN 978-0-470-69978-2 (hardback) – ISBN 978-0-470-69979-9 (paper)
1. Ecology – Mathematics. 2. Ecology – Vocational guidance. 3. Ecology – Research. 4. Quantitative analysts. 5. Quantitative research. 6. Mathematics – Vocational guidance. I. Title.
QH323.5.M387 2011
577.0285'5133 – dc22

2010051191

A catalogue record for this book is available from the British Library.

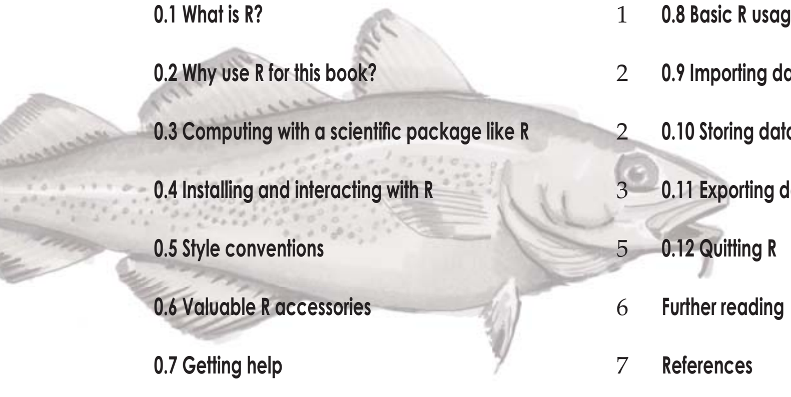
ISBN HB: 978-0-470-69978-2
ISBN PB: 978-0-470-69979-2
ePDF ISBN: 978-1-119-99158-8
oBook ISBN: 978-1-119-99159-5
ePub ISBN: 978-1-119-99172-4

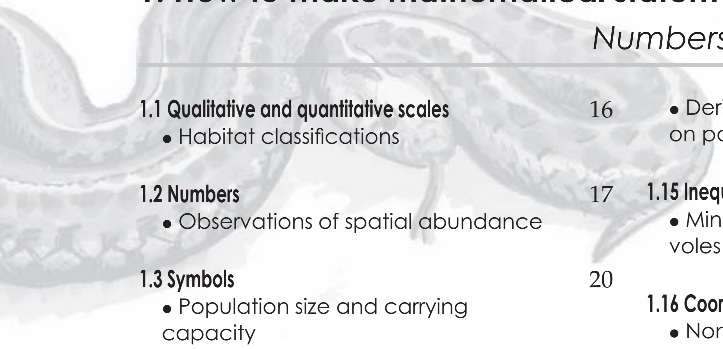
Typeset in 9/11pt PalatinoLinotype by Laserwords Private Limited, Chennai, India

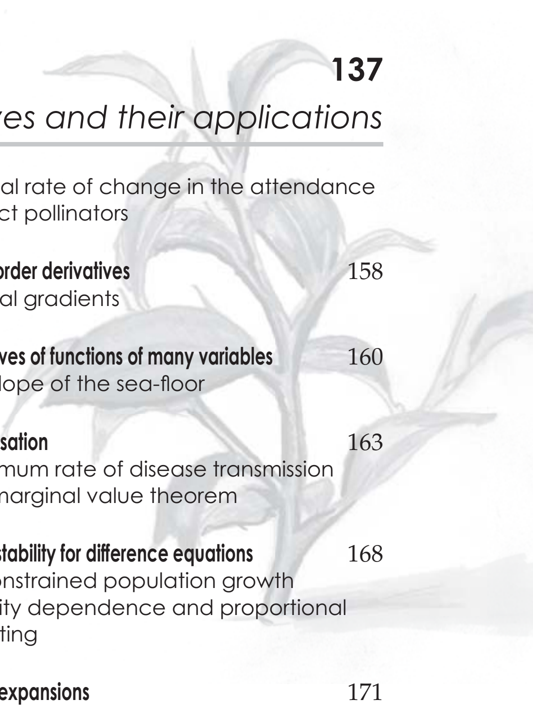
Στους αγαπημένους μου γονείς,
Έφη και Σπύρο

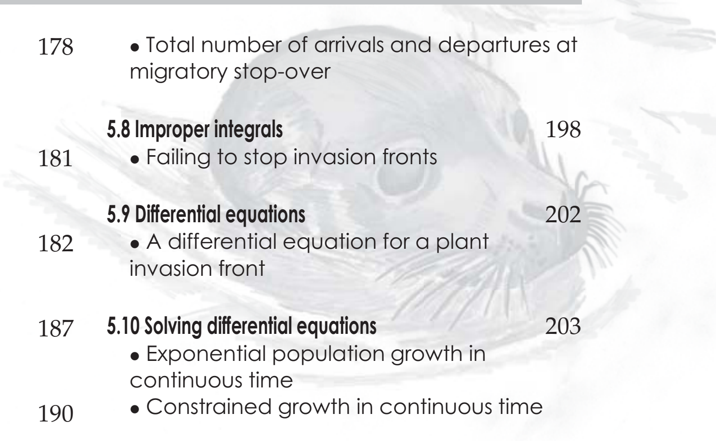
How to be a Quantitative Ecologist

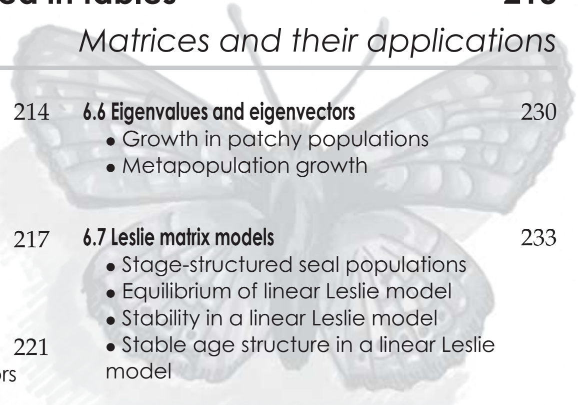
*The **A to R** of green mathematics & statistics*





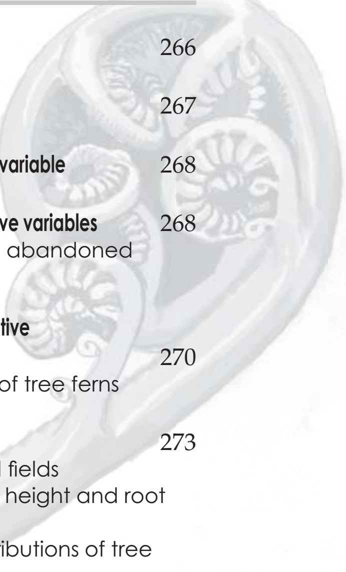






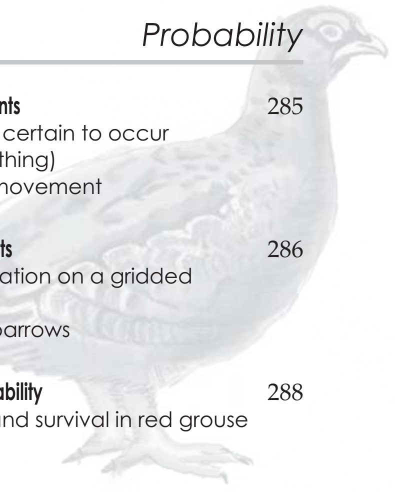
7. How to visualise and summarise data 251

Descriptive statistics



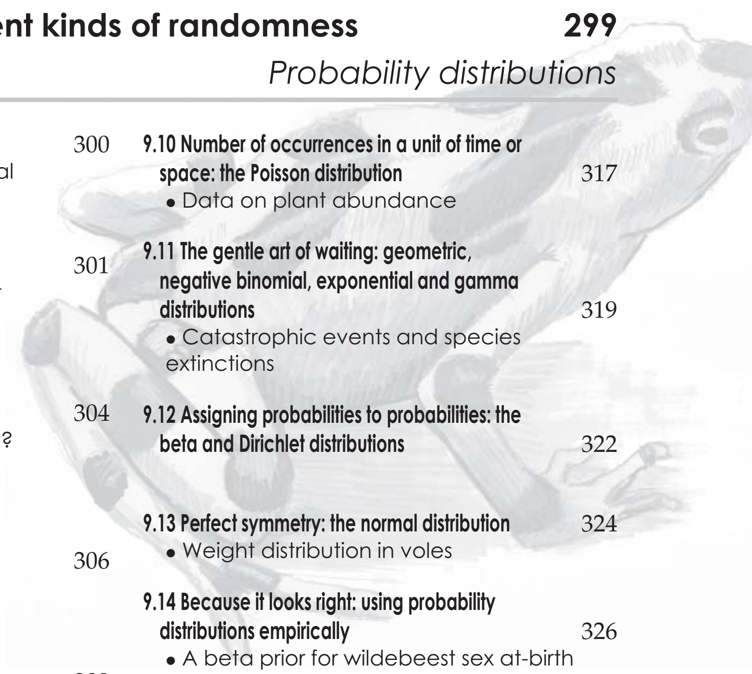
8. How to put a value on uncertainty 279

Probability



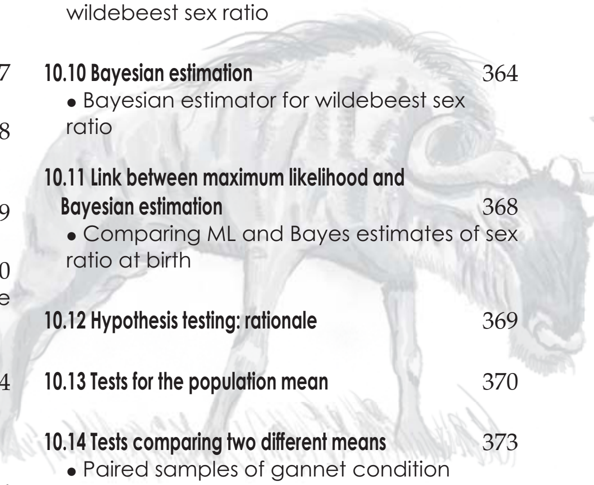
9. How to identify different kinds of randomness 299

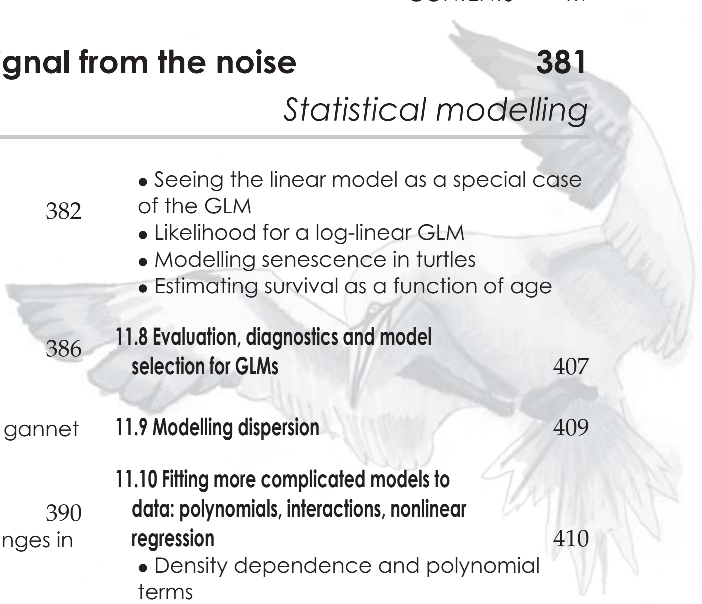
Probability distributions

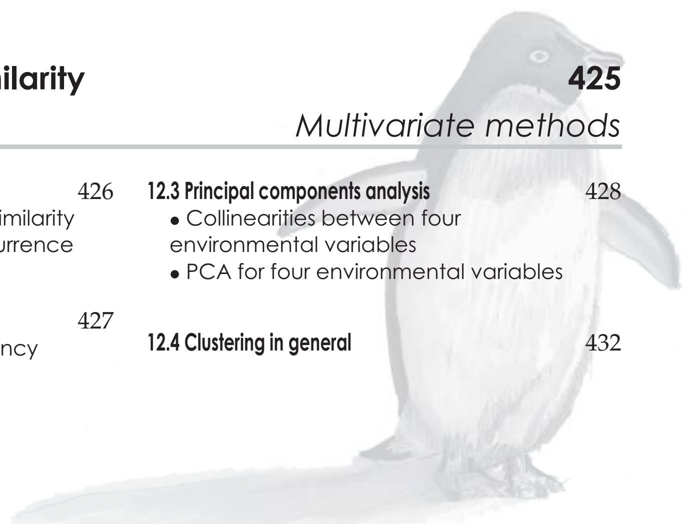


10. How to see the forest from the trees 345

Estimation and testing







How I chose to write this book, and why you might choose to read it
(Preface)

'Many of our biology students are refugees from high-school mathematics'
John Ollason, thinker, cynic and the first quantitative ecologist I ever met.

The evolution of most languages is driven by ease of use and the need for fast information exchange. In this sense, phrases like the 'language of mathematics' and 'computer programming language' are cruel euphemisms (arguably, even seasoned mathematicians find it harder to read through an unfamiliar equation than an unfamiliar piece of text). These languages are driven by the desire to eliminate ambiguity, at the expense of user-friendliness. Because of our excellent ability to perceive context in everyday situations, few of us feel the need to communicate unambiguously. However, scientific research happens outside the comfort zone of well-understood phenomena. For this reason, modern ecologists need to be trained in quantitative methods but find the process painful. Although there is also a great deal of pleasure in using mathematics and computers in science, it is sometimes hard to keep sight of it during the early period of training. Assuming you have decided that you need quantitative skills for your scientific career (a good call), there are five tricks that can make your learning experience less arduous:

❶ Begin from material that you already know well and work your way up to the harder stuff. Try not to be impatient: often, you will discover that you didn't know the basic stuff as well as you thought.

❷ Pick material that contains a good mix of equations and words. The idea of using text around equations is to give an intuitive understanding of their meaning, but you cannot implement the concepts with words alone. So, do not skip the equations. It defeats the purpose.

❸ If you are (or training to be) a research scientist, then quantitative techniques are just a means to an end. So, look for a book that contains lots of examples from your own area of expertise. The mathematical concepts may be the same, but there is no doubting the motivational power of examples that are interesting to you.
❹ You are unlikely to know, in advance, which techniques will come in handy later on in your research. Who knows? Maybe you will need to combine two apparently unrelated techniques to achieve your objective. So, early on, look for breadth rather than depth.
❺ Don't be fazed by notation and terminology. Most terms and symbols have a common-sense, plain-language interpretation. I would hazard that only about 1 in 10 new terms requires repeated readings to digest. If you are struggling with every single term you encounter, you need to go back to a simpler text.

This book is geared towards these requirements. It's the sort of textbook that I wish I had as an ecology student. I have tried to give it a logical structure that nevertheless doesn't make the narrative so linear as to be boring. Each section prepares you for future sections, each chapter builds on previous ones and the entire book prepares you for more advanced texts that are certainly out there. Almost all chapters follow a classical entry to their subject matter and develop to more contemporary themes towards the end. This should re-animate faint high-school skills and then smoothly carry you to what you need to know for your research today.

Focusing on ecology was a selfish choice but it has allowed a happier co-existence between elementary and more advanced material. Basic maths and stats should not be seen as the bitter pill that has to be swallowed in anticipation of the good stuff. Therefore, while the later examples aim to elucidate abstract mathematical concepts by couching them in an ecological context, the earlier examples hopefully show that even basic high-school techniques can be used to address some interesting ecological questions. Indeed, the priority throughout is to convince you that quite a lot of useful quantitative ecology can be done with a modicum of technical knowledge and that some rather fuzzy ecological concepts can readily be recast and understood in formal mathematical language.

Having said that, I have not tried to pretend that quantitative ecology is easy – it is not. Making this admission means that I do not have to skip over the easier bits by pretending that they are trivial and neither do I need to hide the harder bits under the carpet. There are, therefore, few black boxes in the presentation of the theory. I realise that this is a risky decision because most ecologists don't particularly want to know what goes into the methods they are using, but perhaps this is the mentality that we should be working hard to change. As a result, this text contains hundreds of equations, but one of the earlier ones is $1 + 1 = 2$.

The inclusion of computing with R is a double blessing: from your point of view as a reader, it is useful to see how to implement complicated numerical solutions in practice with minimum effort. From my point of view, the extensive R libraries meant that I could include advanced techniques by explaining their theoretical basis but not their exact implementation. This trick makes for a lower page count and stretches some of the chapters towards the cutting edge of the discipline.

It is not the aim of this book to be an exhaustive guide to either R or the science of ecology, but it is most definitely intended as a comprehensive introduction to maths and stats for green scientists. Similarly, I am not hoping to convert ecologists into modellers (although using this book for a structured, two-semester course would go a long way towards this objective). Quite honestly, this material represents the minimum level of quantitative skills currently required of practising ecologists. The choice of topics is broad for two reasons: first, modern ecology is a melting pot of different quantitative concepts and techniques. The best advances in the primary literature seem to come from cross-fertilisation of ideas. Second, the biggest obstacle faced by a neophyte theoretician is psychological: lack of familiarity with the basic terminology and scope of applications means that ecologists lack the confidence to tackle more technical papers and they are left looking for a 'way in' among textbooks that are either far too basic or way too advanced. As a result, many good hunches remain unsatisfactorily verbal (and unpublished) because they are conceived by colleagues who lack the overview of quantitative theory needed to either formalise their own notions or contact the correct specialist. If this book helps a few of these ideas make it into the primary literature, then writing it will have been worth my while. I hope you will find reading it just as rewarding, especially if one of those ideas belongs to you.

Supplementary material for this book (Exercises, Computer projects, R code, etc.) can be found on the online resource : www.wiley.com/go/quantitative_ecologist

Jason Matthiopoulos
St Andrews
20 May 2010

Thank you to . . .

. . .those who toiled

Popi Gkikopoulou, John Harwood, Helen Heyes, Debbie Russell, Gayatri Shanker, Debbie Steele, Students of St Andrews MRes in Environmental Biology and in Marine Mammal Science (classes 2001–2010), Steve Smart.

. . .those who advised

Geert Aarts, Christian Asseburg, Nicole Augustin, Mike Begon, Tim Benton, Luca Borger, Steve Buckland, Peter Corrigan, Will Cresswell, Carl Donovan, Ann Farrow, John Fieberg, Marie Guilpin, John Halley, Sonja Heinrich, Monique Mackenzie, Marc Mangel, Juan Morales, Dave Moretti, Robert Moss, Leslie New, Theoni Photopoulou, Sophie Smout, Matthew Spencer, Simon Wood, Mark Woolhouse.

. . .those who provided

Susan Barclay, Richard Davies, Heather Kay, Ilaria Meliconi, Sheila Russell, Prachi Sinha-Sahay, University of St Andrews, Wiley publishing house, the R development team.

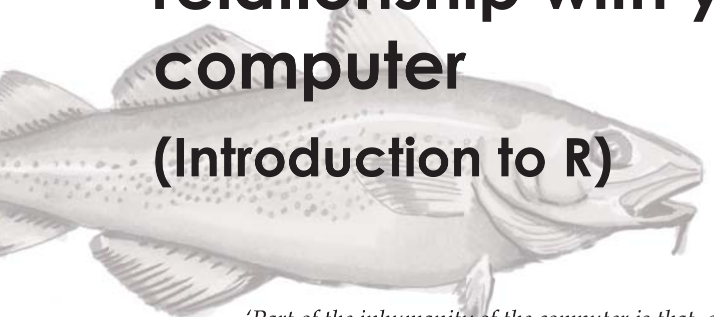
. . .my loved ones

Spyros Matthiopoulos, Spyros Phevos Matthiopoulos, Effie Matthiopoulou, Valia Tavoularie-Matthiopoulou

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.

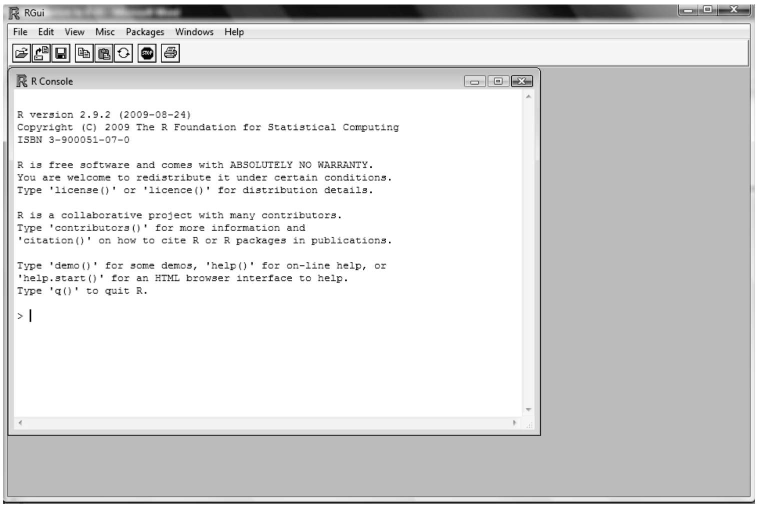
The R base package contains the functionality required by most users including the basic user interface, mathematical, graphical and programming functions and all essential statistical tests and models. It can be downloaded from the Comprehensive R Archive Network (or CRAN for short) at http://cran.r-project.org/. 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
```



```
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.

>
```

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

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.

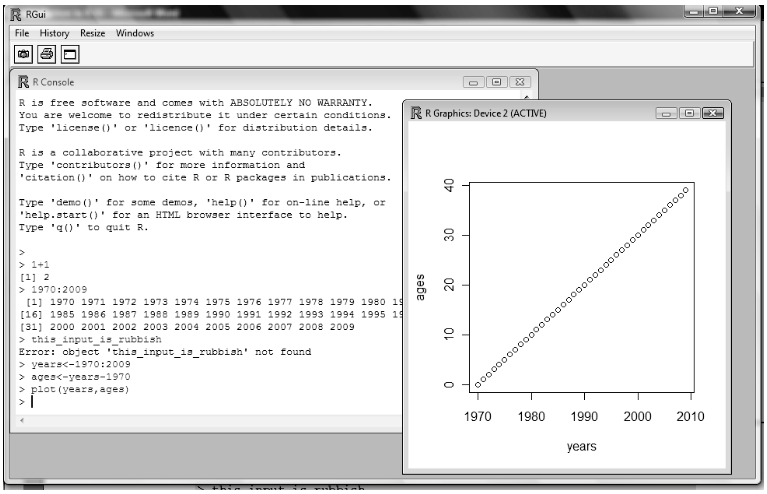
R is **vectorised**, meaning that operations can be applied to entire collections of things with the same ease as applying them to single items. For example, my entire list of birthday anniversaries can be calculated (and filed under the name `ages`) as

```
> 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.



```
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)
> |
```

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

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.