

The Friendly Beginners' R Course for busy people

written by Toby Marthews
at the BCI Research Centre, Panamá (www.stri.org)

This course is only 12 pages long and you work through it in your own time so it's probably the least painful introduction to R currently around. Make sure you have the example files that accompany this text ("first.r", "mystery.r", "quadrats.r" and "quadratdata") otherwise many things won't make sense. Start reading below the line of stars and all should be self-explanatory including how to install R in the first place (if necessary).

Toby, August 2005 (revised October 2006)

SO - you've decided you want to learn to use the R language and environment? Well, hmmmmm ... would it perhaps be more accurate to say that either a) your boss/supervisor/ advisor has told you that you have to and you have a very bad feeling about the whole idea, b) you have some analysis to do and a friend has promised - against all your common sense - that R is easy to use and can help you, c) you have tried reading statistics or modelling books, have given up and are desperately hoping that R is a way around them or d) you've just decided to increase your egg-head rating and impress people?

Whatever your reasons, I think learning to use R *is* a good idea - if only to be aware of what a package like this can do. R does a lot of very clever things and *can* make your life easier if you have to analyse data a lot. The egg-head bit is also a good point: since I put it on my CV everyone believes I'm much cleverer than I really am.

Some comments for those who think R is just another statistics package: Well, R is *both* a programming language *and* a means to do statistical analysis and this is partly why I think it's a step ahead of anything else around at the moment: by learning R you will acquire programming skills (these skills are 70-80% of what people learn (or should learn) in modelling courses) *and* the ability to do statistics on a computer. So, by learning both together, you can gain two sets of skills for the price of one. DON'T BUY AN R TEXTBOOK (at least before you finish this course): firstly because there is a 2300 page R manual downloadable for free from the R website <www.R-project.org> and secondly because R is not 'new statistics' but a way of doing the old statistics quicker so you can and should use a STANDARD statistics textbook, just adding notes to it as required.

I've used R for only about a year which means I really don't know the ins and outs of it, but in the following few pages I should be able to give you a kick-start and that'll be enough for you to be able to write your own R scripts, use some R functions, draw some nice graphs and generally get familiar with it. This guide is written for someone who's

used a computer before but has NO PROGRAMMING EXPERIENCE (if you do have some experience, you'll know which sections to skip below).

I can't say how long this text will take to work through (everybody's different), but there are only 6 challenges so hopefully not too long. Set yourself up with a computer, a printout of this text, a strong coffee (or alternative stimulant) and go through the sections one-by-one starting with

Installing R & Running an R Program

You need a bit of general knowledge of computers and how they work first. If you already know about computer languages and workspace directories and have R installed on your computer then go on to the next section.

Computer programs are always written in some kind of computer language. Computer languages are either script ones (e.g. BASIC, JavaScript, R) or compiled ones (e.g. FORTRAN, PASCAL, C++, Java) and whichever one a programmer is using, it all has to be translated in to machine code (which is a stream of 1s and 0s) before the computer can actually 'execute' or 'run' it (= do it). Here's where the difference lies: with script languages the computer goes through the program line-by-line and translates and executes each before going on to the next line; with compiled languages the computer translates the whole program in one go, saves the machine code as an 'executable' on disk and then runs the executable directly.

Generally speaking, script languages are *slow* but *more user-friendly* (esp. *error-reporting*) and compiled languages are *much faster* but *have less straight-forward syntax*. So, if you write a program in R then it'll run a lot slower than an equivalent program written in C++ - and you should be aware of this - BUT a) the difference will only be noticeable to you if you're doing really lots of calculations, b) if you've never used a computer language before then you'll be pulling your hair out if you start with something like C++, c) in the case of R there are all these extra features like graph-plotting and statistical functions that can make your life a LOT easier (and FORTRAN, for example, can't do those without special add-ins) and d) if you learn how to *program* using a language like R then you'll find it really easy to pick up any other computer language *afterwards* because all languages have similar structures (repeat loops, for loops, if statements, etc.).

That's all just to set the scene: let's actually do something. Here's how to install R on your computer. I've done instructions here for WINDOWS and for LINUX (I don't know anything about Apple Macs although R is available for that too) that work at the time of writing for my machine and therefore *should* work fine for you too:

INSTALLATION FOR WINDOWS:

1. Go to the R website <www.R-project.org>, click on Download/CRAN on the left and choose a mirror site near you (any will do). Choose Windows and click on “base”, download the Setup Program (it’ll have a name ending in “.exe” like “R-2.4.0-win32.exe”) and save it on the Desktop. Double-click on this program to run the installation (make sure you tick the options to get all the Help Pages and accept the default startup options). Yes to a Start Menu folder, but NO to a desktop icon and NO to a Quick launch icon (see Step 3). R is now on your computer (and you can delete the “.exe” file).

2. Create a workspace directory on the Desktop (or elsewhere if you prefer) for using R (right-click on the Desktop background, choose New -> Folder and give it a name like “rwork”) and copy “first.r” (accompanying this text) in to it. This directory is used by R for storing variables and function definitions (in a file called “.RData”) so you have to have one. Oh, and “A -> B” is my way of saying “go to menu A and select B from it”. WATCH OUT: in a particularly annoying way, some windows systems automatically rename email attachments called “xxx.r” as “xxx.r.txt” or “XXX.R.TXT” when you save them and you need to keep renaming them back to “xxx.r”.

3. Open the “RGui” by going through the Start Menu (something like Start -> Programs -> R -> R 2.4.0) - by the way “Gui” = “Graphical User Interface”. Use “Change dir...” in the File menu to change directory to the one you created in Step 2. Now use “Save Workspace...” in the File menu to create an “.RData” file in the workspace directory and please change the default save name of “.RData” to “start.RData”. Close R (click on the red “X” in the top right and you don’t need to save the workspace). Now you’re back in normal windows. Open the directory you created in Step 2 and create a shortcut to “start.RData” (right-click on it and choose “Create shortcut”). Now rename that shortcut “Start R” and move it to the Desktop. The point of all this was to make an icon on the desktop that you can double-click on to start R (the one R would create at installation starts up with the wrong workspace directory). If you want a Quick launch icon on the task bar as well, use the mouse to drag “Start R” on to the task bar (normally just to the right of where the “start” of the Start Menu is).

4. Now start up R (i.e. double-click on the “Start R” shortcut). Test R can run a simple program: use “Source R Code...” in the File menu, find first.r in the workspace directory and open it. R will run the program and you should get a welcome message (the file first.r is just a text file, by the way, as you can see if you open it in WordPad).

5. Not quite finished yet: go to File -> Open script... and choose first.r. An R Editor window should open up to allow you to change the program (I need to check you can do this too). Find the “5” on line 6 and change it to a “10”. Save it by going File -> Save as... and save it under the name “first2.r” (then close the editor window).

6. Now run first2.r in the same way as in Step 4. If you got 10 stars then you’re doing well and you deserve them!

7. You can exit R by clicking on the red “X” or by typing “q()” (for now, you don’t need to save the workspace image again).

INSTALLATION FOR LINUX:

1. Go to the R website <www.R-project.org>, click on Download/CRAN on the left and choose a mirror site near you (any will do).
2. Choose Linux and find the right download file for your version of Linux (e.g. .rpm) and then install it in the way your version of Linux expects (you should know what way - probably either with a double-click or through something like YAST).
3. Create a workspace directory on the Desktop (or elsewhere if you prefer) for using R and copy “first.r” (accompanying this text) in to it. This directory is used by R for storing variables and function definitions (in a file called “.RData”) so you have to have one.
4. Open a terminal, change directory in to your workspace directory using cd and type “R” to go into the R language (the prompt will change to “>”).
5. Test R can run a simple program: type “source(“first.r”)”. R will run the program and you should get a welcome message (the file first.r is just a text file, by the way, as you can see if you open it in a text editor like GNUemacs, kate, gedit, ue, pico, vi, etc.).
6. Not quite finished yet: open first.r in a text editor (NOT using the terminal - leave that open at the same time and do this in a different window) so that you can change the program (I need to check you can do this too). Find the “5” on line 6 and change it to a “10”. Save it under the name “first2.r” (then close the editor window).
7. Now run first2.r in the same way as in Step 5. If you got 10 stars then you’re doing well and you deserve them!
8. You can exit R by typing “q()”. For now, you don’t need to save the workspace image.

Two Windows: Console & Editor

With the heady feeling of success from having run your first R script, I'm sure you'll be wanting more, more, *more!* Well, just to get you used to what we've done up to now, please could you open up the original first.r in to your editor again. See if you can manage to do the following two things:

Q1. Can you make the FOR loop count *down* from 5 to 1 instead of up from 1 to 5?

Q2. Can you make it count up and then down (which is easiest to do using two FOR loops one after the other)?

If you try those two questions (I know they're tedious: you've got to learn to walk before you can run) then you'll have to get used to the way R programmers keep two windows open at once: you edit the program in an "editor" window, then save it, flip to the "console" window (aka. "terminal") and run the program from there (Windows version only: note the different "File" menus depending on which window is active). This is the way programming is done in a lot of languages, by the way, and many people resize and move the two windows so they are as large as possible without overlapping. Windows users may get annoyed about the way you can't get the editor window to 'word wrap' (at least I can't see how to) and choose to use WordPad for all editing (saving in text format each time). Whichever editor you use, make sure it always tells you what line number you're on (excl. continuation lines) so that you can use R's error reports (Windows users: I'd recommend strongly using the text editor "TextPad" (www.textpad.com and you can download an evaluation version to try out for free) which is *very* much nicer to use than either NotePad, WordPad or R's in-built text editor).

Please don't skip Q1 and Q2: they're there to force you to check that the editing-saving-running process works OK on your version of R and you need this to be working for what follows. If it doesn't work then please re-check what you've done so far and/or panic and call for help (try the FAQs about installation on www.R-project.org). A "syntax error", by the way, means there's something wrong in the code you're editing: check for typos and unclosed brackets and things like that.

The R Manual

While you're concentrating on `first.r` to answer those questions, please make sure you can understand what *every* line does. I haven't explained everything in my comments there (the # lines) because you need to get into the habit of using R's very comprehensive manual system. There are no annoying paperclips, funny dogs or wizards: just lots of information. Here's how to use it:

Imagine you can't work out what the "cat" command does (yes, I know this one is probably obvious, but bear with me ...): go to the Console window and type in "?cat". The manual 'page' for cat will then appear (in Windows it appears in a new window, in Linux in the same window: you press "q" to go back to normal). These manual pages are written in a pretty technical way (you're going to get used to it, I'm afraid) BUT you generally don't have to read much of it: scroll down to the bottom and there are usually some helpful examples and these are the most useful bit of the page to start with because you can copy them in to the Console window to see what they do (in Windows mark the example you want with the mouse, do CTRL+c to copy, click on the Console and do CTRL+v to paste; in Linux mark it and do Edit -> Copy, then q, then Edit -> Paste). Sometimes these examples try to be a bit too clever and it's not clear how they work (like the two on ?cat, I think, which produce "iteration = ..." and "{1}: a 100 b 200 ..."), but for many commands the examples are very useful, e.g.

Q3 Copy and try out the "Discrete Distribution Plot" example at the end of the "plot" manual page and the "setting row and column names" example from the "matrix" manual page.

If there doesn't appear to be a manual page for a particular command (e.g. try typing "?for"), there is a search facility you can use: type "help.search("for")" and top of the results list is "Control(base)" which is a page you can bring up by typing "?Control" (note the capital "C"). Perhaps a more user-friendly way of searching for help is to download the "R Reference Index" from the "Manuals" part of the R website (www.R-project.org): this is in PDF format and you can search for words in it using CTRL+f.

These search facilities are also very useful for finding out how to do things on R, e.g. a standard kind of statistics plot is a box plot, but at the moment you don't know how to do this in R and if you type "?box" you don't get the right manual page. Typing "help.search("box")" in to the Console Window, however, or searching for "box" in the reference index will both lead you to the keyword "boxplot" which is the right one to use (and both sources give you examples to try too).

Putting Commands Straight in to the Console Window

I hope you like these short, easily-digestible sections by the way: I'm trying only to tell you what you *need* to know to use R. Just to make sure everyone is following, I'd better give the answers to Q1, Q2 & Q3: for Q1 just change the "1:5" to "5:1", for Q2 you have two loops with the first going up (1:5) and the second going down (4:1 to avoid having 5 counted twice), for Q3 you should get a pretty graph ("rpois(100,lambda=5)" means 100 draws from a Poi(lambda=5) distribution - we'll get to this sort of thing later) and a 2x3 matrix with 1,2,3 on the top row and 11,12,13 on the bottom row. All those who got these answers get 10 stars (are you keeping track of your stars?).

Next, please click on the console window and type in "y=3" and ENTER. Now type "y" and ENTER. Now type "y=y*20" and "y" again. Do you see what's going on? You can put commands in straight like this. Now type "cat("Free love starts at",y,"\\n")" and "for (i in -4:2) {". The prompt has changed from ">" to "+", which means R has found an incomplete command and you need to type more in, so type "cat(i,"\\n")" followed by "}". You get the idea, I think. Type "ages=c(13,41,49,0,42,1,40,20)" followed by "hist(ages)" to get a quick taster of R's statistical side. Also, there is a "history" function whereby you can press the up and down arrows to find, modify and re-use a previous command: click back on the Console Window and press the up arrow a few times to get the first "cat" command in this section, change "love" to "dental care" and press ENTER to get "Free dental care starts at 60" which is, of course, what I meant to say really.

This facility of being able to try out any command directly is really powerful and one of the main reasons for using a script language (you can't do it so easily with compiled languages). If you're given a program containing lots of incomprehensible command lines (as may very well happen in the next section...), you can try out the lines one-by-one by copying them in to the Console Window and seeing what they do.

A Mystery Program

Time for another challenge. Have a look at the program `mystery.r` (accompanying this text): your task is to work out what it does.

When trying to work out what a program does, the first thing to do is to run it (save it in the workspace, start R, use source, etc.). The second thing to do is to open it in the editor and see if you can figure out from the code and code-comments what it's trying to do. You'll get another 15 stars if you can tell me:

Q4 What does the program `mystery.r` calculate and put in the NND column of its printout at the end?

At this point many people might be saying “whoa – what?” because we've suddenly jumped to something with `sin`, `cos`, `abs`, `sqrt` and `pi` in it (maths, I'm afraid), a couple of control loops (the `for (...) { ... }` bits) and arrays (the variables with something in `[]` after them). Don't panic (yet) – just find any commands you're not familiar with, check the manual pages to see what the keywords are supposed to do (e.g. type “`?sqrt`” and copy the example at the end in to the Console Window to see what it does) and, if that doesn't help, copy the whole line you're not sure about in to the Console Window to see if it works there too (most will), alright?

Q5 By inserting “`f=jpeg(file="plot1.jpg")`” just before the `plot` command and inserting “`dev.off()`” just after the `abline` commands, the plot should appear in the workspace directory as a `.jpg` graphics file instead of being plotted on the screen. Type “`?jpeg`” to find out how to export as a `.bmp` or `.png` file too. This is very useful because it allows you to put plots into a Word document or something similar. Another 5 stars if you manage this too.

So, how are you feeling about this gentle art of programming so far? Dead hard or insultingly easy? Keep count of your stars and pat yourself on the back if you got this far. Take off a star for every bit of help you got from someone else, by the way: I'm watching you.

The answer to Q4 is at the end of this text, written backwards, but DON'T LOOK NOW: work it out first!

Vectors

OK, I think you need a breather!

This is an easy section just to tell you more about “vector” data types, which are the `c(...)` lists of numbers you've already met. Click on the Console Window and type “`b=10;c=11;d=12`” (the semi-colon is just a way of putting more than one command on one line). Now type “`vec=c(3,b,b,8)`” and “`vec`” and you can see that `vec` now holds a list (aka. vector) of numbers. You can access the numbers directly too: type “`vec[2]`” and “`vec[4]`”

and “for (i in 1:4) {cat(vec[i],"\n")}}” (nb. 3 different types of brackets and they have to be in the right places).

Now type “vec2=c(1,2,3,4)” and “vec3=vec+vec2” and “vec3” and “vec4=(vec*vec2)+6” and “vec4” and “vec5=c(rep(3,times=20))” and “vec5” and if you can follow what’s going on with all that then you know pretty much all you need to know about vectors.

Matrices

After vectors come matrices. Boringly, this is not the enslaving-the-human-race, world-simulating, karate-kicking kind of matrix (the R people have not implemented that yet), but a more traditional mathematical matrix, which is just a grid or array of numbers. Type “mat=matrix(c(7,8,2,3,4,5,6,3,2,1,-2,-9),nrow=3,ncol=4)” and “mat”, which should give you a matrix. Now type “mat2=matrix(c(7,8,2,3,4,5,6,3,2,1,-2,-9),nrow=3,ncol=4,byrow=TRUE)” and “mat2” and “mat3=matrix(7,nrow=3,ncol=4)” and “mat3” and “mat4=matrix(3:15,nrow=3,ncol=4)” and “mat4” and “mat2+mat3” and “mat4*2” and phew, but I think we’ll stop there!

All these commands about vectors and matrices can be used in a proper program in just the same way as they work in the Console Window. If you feel that hell would freeze over before you voluntarily did anything with matrices then fair enough: we’ll have no more of them here.

Plotting Data on a Graph

R has a lot of funky plotting functions and these are one of the main reasons for learning how to use the thing SO here are a couple of examples. Try typing the following in to the Console Window to get a basic plot:

```
years=c(2004,2005,2006,2007,2008)
rainfall=c(1500,1300,1800,1350,1950)
plot(x=years,y=rainfall)
```

Now that’s fine, but the plot comes out using the R defaults which may not be to everyone’s taste and they also don’t correspond with the standard guidelines on figures that most scientific journals insist on (e.g. see “Figures” on the Journal of Ecology page <www.blackwellpublishing.com/submit.asp?ref=0022-0477>). If you look at the manual pages ?plot and ?par you’ll find out how to change some of the formatting options and here is an example of a different way of displaying the same data:

```
thisdata=data.frame(years,rainfall)
bestfit=lm(rainfall~years,data=thisdata)
plot(thisdata,main="Annual Rainfall",xlab="recorded
years",ylab="mm",bty="l",ylim=c(0,2000),pch=4,sub=paste("Best Fit line is y =
(",bestfit$coefficients[2],") x + (",bestfit$coefficients[1],")"))
lines(thisdata)
abline(bestfit,lty=2)
```

Lastly (for this section), some bright spark at the R office decided to write a nice graphics demonstration for people like you and me and you should definitely have a look: type `“demo(graphics)”` and keep pressing ENTER or RETURN to go through it.

Remember to look at *all* the manual pages of *all* these various plotting commands, by the way, and try the examples: that’s the only way you can learn how to make all those impressive plots.

Packages

For a lot of statistical analyses you have to get to know R’s system of ‘packages’. Type `“library()”` in to a Console Window to find out what packages were installed on your computer when you put in R. Now type `“search()”` to get a list of the packages from that list that are already loaded in (installed packages are not necessarily loaded in, you see).

A package like `“methods”` is already installed and loaded in by default, but `“survival”` is only installed so if you want to use it you have to load it in by typing `“library(survival)”` or `“require(survival)”`. Try the command `“date.mdy(sdate=15000)”` both before and after loading in survival (which should give you 25-JAN-2001) to see what I mean.

If you want to use a package that isn’t already installed, then, hmmm To use an R function that’s not in one of the installed packages you’d have to be doing something pretty abstruse To be honest, if you’re reading a beginners’ course on R (which you are) then you should have no cause whatsoever to be using uninstalled packages

OK, if you insist, then go to www-r-project.org, click CRAN on the left, choose a mirror site near you and then click Packages on the left and start reading up on how to do this (I’m not going through this in a beginners’ course!).

Quadrat-o-phenia

This is only supposed to be a beginners' course and I think it's long enough already so I'm going to wrap it up here by giving you a final example program that demonstrates some R features (vector arithmetic, reading in data from a textfile, a χ^2 test, a bar plot, a function definition and sampling from a Poisson distribution). Have a look at `quadrats.r` (accompanying this text) and try to figure out how it works using the manual resources I've been telling you about above (the program explains what it's working out as it goes along). Apart from running this program and pausing in admiration at the clear and concise way I write code (or perhaps staring in disbelief at how much pain can be compressed in to a single page of text ...), I'd like you to do something as a final challenge.

Q6 Dig out a *different* worked example of a χ^2 test (you may have encountered one before in a textbook or you could look at a webpage like www.mste.uiuc.edu/patel/chisquare/intro.html). They should all follow exactly the same calculations and I'd like you to modify the program `quadrats.r` so that it calculates whatever problem you've dug out and gets *the same answer* as your book or website. A big, fat 30 stars to you if you can do that!

Whilst trawling through `quadrats.r` you should have found examples of many unfamiliar commands. This is deliberate and you may have noticed that I've been trying to slip in to all these programs as many different useful commands as I could. This is because I want you to keep the programs and use them.

You see, no-one ever really writes an R script straight off from scratch: most programmers maintain a little 'library of useful programs' somewhere and when they have to write a new one they start with an old program in this library and modify it until it can do the job in hand. There's nothing bad in this (C and FORTRAN programmers have even published many standard routines - see "Numerical Recipes" on www.nr.com - and maybe R will have something similar one day) and the idea of giving you these scripts is that they will form the kernel of a similar library for you.

All Done and Dusted

That's it! The end of the course.

For further information, the R website <www.R-project.org> has links to a lot of different things on it including a newsletter, a mailing list for help and MANUALS IN CHINESE, CROATIAN, FRENCH, GERMAN, HUNGARIAN, ITALIAN, JAPANESE, SPANISH, POLISH AND PORTUGUESE (see Documentation/Other -> Contributed Documentation on the website). If you've got a problem and you've tried all my suggestions in this text then email this mailing list.

For each star you managed to earn during this course (max. 70), please now go out and buy a seed or a plant and put it in your garden or somewhere around wherever you're living (native species only, of course). In this way the study of R can contribute to the greening and beautifying of the world outside, which makes me happy because I work with plants.

Any CONSTRUCTIVE AND NOT ABUSIVE comments you want to make about this little course, you can send to <raspberry@webmail.co.za> if you really have to, BUT I only read this about once every 6 months so please don't expect answers to any questions. The whole point of this course is that you work these things out yourself, I'm afraid!

Ciao,
Toby.

PS. Most R users (and most R manuals) say you should type "`x <- 3`" instead of "`x=3`" to make x equal 3. There are reasons for this, but I find "`x=3`" much more straight-forward (and certainly easier to explain to my boss when he looks at my code!) and they are both equivalent anyway.

PPS. My three most common errors when writing R code, are

- i) forgetting that "`log(x)`" means $\ln(x)$ not $\log_{10}(x)$,
 - ii) forgetting to use "`==`" in if statements (e.g. you have to type "`if (x==3) { ...`" instead of "`if (x=3) { ...`") and
 - iii) comparing with a negative number (e.g. "`if (x<-3) { ...`" doesn't work - you have to say "`if (x<(-3)) { ...`").
- Watch out for these.

PPPS. Almost forgot: the answer to Q4 is: "ecnatsid ruobhgien tseraen" rof sdnats "DNN"