Part A Simulation and Statistical Programming HT14

Lecturer: Geoff Nicholls

University of Oxford

Lecture 3: Statistical Programming and R: an introduction

Notes and Problem sheets are available at

www.stats.ox.ac.uk\ ∼nicholls\PartASSP

Today: Statistical Programming in R

- an introduction to basic use of R
- some examples from the simulation part of the course
- 1 hour of me doing R demos and lecturing
- 1 hour of you doing a practical at the computers

R is an open-source package for Statistical Computing

It is freely available - `http://cran.r-project.org/`

What is Statistical Programming?

There are three elements: statistics, algorithms and carrying out statistical analysis on a computer.

In a typical study we begin by loading the data into the computer, and making an exploratory analysis of the data (EDA).

We look at the raw numbers, and visualize them by plotting graphs that reveal patterns in the data. We may have a model we wish to fit to the data, and a hypothesis we wish to test.

Statistics tells us what objects we need to calculate in order to make the fit and carry out the test.

An algorithm tells us how to calculate those objects. It is a sequence of operations that carry out a given task. An algorithm is a mathematical object, with properties we can prove. We give an (informally stated) algorithm to convert a decimal to binary later in this lecture.

Finally we go back to the computer and implement the algorithm (or use an implementation someone else has written). This is a computer programme and it should carry out the calculations dictated by the algorithm. Two correct implementations of the same algorithm can differ, for example in efficiency and in the range of cases they handle correctly.

Getting started

Begin by selecting `Start` $>$ `all programmes` $>$ `R` The R Graphical User Interface **RGui** opens, with an internal window, the `R Console`, into which you type commands.

You will often want to keep a record of your work from an R session. You can do this by starting an R script: `File > New Script`. This opens a new file in an R editor window.

You can type R commands in the script, which can then be selected (using the mouse) and evaluated (using Ctrl-R) in the R Console.

You also make notes in the script and save the script for the future `File > Save` (you need to select the R editor window for the script you want to save using the mouse).

Arithmetic Operators

- Arithmetic operators: `+, -, /, *, ^` where `x^y` is x^y.
- `%%` for modulo reduction, `%/%` for integer division, `%*%` for matrix multiplication

Example

- Type `2 + 3 <return>`
- Type `6 * 9 + 3 / 5 <return>`
- Type `3^4 <return>`
- Type `5^(-5.6+3) <return>`
- Type `143 %% 6 <return>`

Mathematical functions

Many mathematical functions are available e.g.

```
exp(x), log(x), log10(x), sqrt(x)
sin(x), cos(x), tan(x)
asin(x), acos(x), atan(x)
sinh(x), cosh(x), tanh(x)
asinh(x), acosh(x), atanh(x)
abs(x), min(x), max(x)
```

- Type `exp(-4 * 4 / 2) / sqrt(2 * pi)`**<return>**

- Type `log(4)`**<return>**

Functions arguements

Functions can have more than one argument. These can identified by their names, or by order.

- Type `choose(6,3)<return>`

- Type `choose(3,6)<return>`

- Type `choose(n=6,k=3)<return>`

- Type `choose(k=3,n=6)<return>`

The first two differ, the last two give the same answer.

Functions for familiar distributions

`rDBN(),dDBN(),pDBN(),qDBN()` are random numbers, the pdf/pmf, the cdf and quantiles for the distribution `DBN`.

`rexp(n=10,rate=1)` gives 10 Exp(1) rv.

`pnorm(q,mean=0,sd=1)` is $P(Z \leq q)$ for $Z \sim N(0,1)$

`dpois(x,lambda)` is $\exp(-\lambda)\lambda^x/x!$

If `q=qt(p=0.975,df=4)` then $0.975 = P(X \leq q),\ X \sim t(4)$.

What does this return? `rchisq(n=10,df=5)`

Getting Help

- `help(command)` or `?command` if you know the command name

- Type `help.search("subject")` or `??subject` for a subject

- `?‘&&‘` for operators or words such as `if`

- `help.start()` for R online documentation

- Help menu in Windows gives still more options

Vectors and sequences Vectors or sequences of numbers can also be created, stored and manipulated

- They can be created manually
  `x1 = c(0.1, 0.2, 0.3, 0.4, 0.5)` **`<return>`**

- Functions apply to vectors element by element

  Example

if `x = (1, 2, 3, 4)` and `y = (5, 6)`, then

```
x + 3 = (4, 5, 6, 7)
x + y = (6, 8, 8, 10)
x^2+max(x)=(5,8,13,20)
```

Logical Operators

- `==` (equal), `!=` (not equal), `>`, `<`, `>=`, `<=`

  Example
  ```
  3 < 4 <return>
  c(0.1,0.3,0.6) > 0.23 <return>
  ```

- `!` (not), `|` (or), `||` (or), `&` (and) `&&` (and)

- Note difference between `|`,`&` and `||`,`&&`: former work on vectors, latter (a) only consider first element and (b) stop as soon as the outcome is known.

Example (a)

if `x = (TRUE, FALSE, TRUE)`
and `y = (FALSE, TRUE, TRUE)`
`x | y = (TRUE, TRUE, TRUE), x || y = (TRUE)`, then
`x & y = (FALSE, FALSE, TRUE), x && y = (FALSE)`

Example (b)

If `b=2` and we execute `(a=1)<0 & (b=-1)<0` then `b=-1` but if `b=2` and we execute `(a=1)<0 && (b=-1)<0` then `b=2` (still). In the first case we have a sideffect (`b` changes value). It is usually a bad idea to allow them in your code so we would not typically include an assignment `b=-1` in a test `(b=-1)<0` irrespective of whether we used `&` or `&&`. The `&&` form saves time evaluating later tests when the result is already known.

Error messages

- The R commands you enter will sometimes contain errors.
- R will either report an Error message or prompt you to complete the command.

Example
Incomplete brackets i.e. `(4+8`**`<return>`**
prints a + which means you need to enter the remaining brackets.

Example
Too many brackets i.e. `(4+8))`**`<return>`**
results in an error message.

Example
Using the wrong type of brackets i.e. `exp{4}`**`<return>`**
results in an error message.

Computer representation of numbers

- On a computer numbers are stored in binary rather than decimal.

- Consider the number 1197.625.
  In decimal we write

10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}
1	1	9	7	6	2	5

  In binary, the number is

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1	0	0	1	0	1	0	1	1	0	1	1	0	1

To convert from a base-10 integer to base-2 (binary), the number is divided by two, and the remainder is the least-significant bit. The (integer) result is again divided by two, its remainder is the next most significant bit. This process repeats until the result of further division becomes zero.

$$1197 = 2 * 598 + 1 \longrightarrow 1$$

$$598 = 2 * 299 + 0 \longrightarrow 0$$

$$299 = 2 * 149 + 1 \longrightarrow 1$$

$$149 = 2 * 74 + 1 \longrightarrow 1$$

$$74 = 2 * 37 + 0 \longrightarrow 0$$

$$37 = 2 * 18 + 1 \longrightarrow 1$$

$$18 = 2 * 9 + 0 \longrightarrow 0$$

$$9 = 2 * 4 + 1 \longrightarrow 1$$

$$4 = 2 * 2 + 0 \longrightarrow 0$$

$$2 = 2 * 1 + 0 \longrightarrow 0$$

$$1 = 2 * 0 + 1 \longrightarrow 1 \quad \longrightarrow 10010101101$$

To convert from a base-10 fraction to base-2 (binary), the number is multiplied by two, if the result is $>$ 1 the most-significant bit is 1 otherwise 0. The (fractional) remainder is again multiplied by two and compared to 1. This process repeats until the remainder is zero.

$$
\begin{aligned}
&0.625 * 2 = 1.25 \rightarrow \text{1} \\
&0.25 * 2 = 0.5 \rightarrow \text{0} \\
&0.5 * 2 = 1 \rightarrow \text{1} \qquad \rightarrow \text{.101}
\end{aligned}
$$

So, $1197.625_{10} = 10010101101.101_2$

- In decimals, some fractions are recurring. The same can be true in binary. Consider 0.8

$$\begin{aligned} 0.8 * 2 &= 1.6 \longrightarrow 1 \\ 0.6 * 2 &= 1.2 \longrightarrow 1 \\ 0.2 * 2 &= 0.4 \longrightarrow 0 \\ 0.4 * 2 &= 0.8 \longrightarrow 0 \end{aligned}$$

  etc

  So, $0.8_{10} = 0.\overline{1100}_2$

- Fractions in binary only terminate if the denominator has 2 as the only prime factor.

- Modern computers use 64 bits to represent numbers so some numbers (like 0.8) must be approximated.

- Care is needed when testing whether 2 numbers are the same:

    - `x <- seq(0, 0.5, 0.1)` ##generate a sequence from 0 to 0.5 in steps of 0.1
    - type `x` ##Look at x
    - is `x` `equal to (0, 0.1, 0.2, 0.3, 0.4, 0.5)`?
    - To find out, type
      `x==c(0, 0.1, 0.2, 0.3, 0.4, 0.5)`

```
[1] TRUE TRUE TRUE FALSE TRUE TRUE
```

Rounding problems Tiny inaccuracies can accumulate:

- the sample variance of a vector `x` is often calculated as $var(x) = (\sum x^2 - n\bar{x}^2)/(n-1)$ or, in R
  ```
  (sum(x^2) - n * mean(x)^2) / (n - 1)
  ```
- Try it with `x <- seq(1:100)`
- Now with `x <- seq(1:100) + 10000000000`
- compare with `var(x)`
- Can you see why there was a problem?

Moral "Worry, but, if you use R, don't worry too much, because R has worried for you."

Try it yourself

Begin by selecting `Start` > `all programmes` > `R`

The R Graphical User Interface `RGui` opens, with an internal window, the `R Console`, into which you type commands.

You can open the lecture script: `File > Open Script`. The script you want is `L1.R` on the `H:` drive.

After 5 minutes we will continue with the lecture.

Basic Types of Variables Variables are the equivalent of memories in your calculator. But you can have unlimited (almost!) quantities of them and they have names of your choosing. And different types. The basic types are

- integer
- double
- character
- logical: these take one of the two values `TRUE` or `FALSE` (or `NA`, see later)
- factor or categorical

More Specialised Classes

As well as the basic types of variables, R recognises many more complicated objects such as

- vectors, matrices, arrays: groups of objects all of the same type. The practical will show you how to use these.

- lists of other objects which may be of different types

- Specialised objects such as Linear Model fits

Special values of objects

There are some types of data which need to be treated specially in calculations:

- `NA` The value `NA` is given to any data which R knows to be missing. This is not a character string, so a character string with value '`NA`' will be treated differently from one with the value `NA`.

- `Inf` The result of e.g. dividing any non-zero number by zero

- `NaN` The result of the operation isnt defined (eg `0/0`, `Inf-Inf`, `log(-1)`)

Factors: Factors are variables which can only take one of a finite set of discrete values. They naturally occur as vectors, and can be

- numeric e.g. drug doses with values 1mg, 2mg, 5 mg
- or character e.g. voting intention with values Liberal Democrat, Conservative, Labour, Other

Although factors are stored as numbers, along with the label corresponding to each number, they cannot be treated as numeric. Would it make sense to ask R to calculate **`mean(voting intention)`**?

A more useful function for factors is **`table`** which will count how many of each value occur in the vector.

More about Factors

- Ordered Factors

  Some factor variables have a natural ordering. Drug doses do, but voting intentions usually do not. R will treat the two types differently. It is important not to allow R to treat non-ordered factors as ordered ones, since the results could be meaningless.

- Creating factors

  Use `cut` to create factor variables from continuous ones:

Example

```
age <- runif(100) * 50
table(cut(age, c(0, 10, 20, 30, 40, 50)))

 (0, 10] (10, 20] (20, 30] (30, 40] (40, 50]
     17       19       19       24       21}
```

Data frames

- For storing data which is a collection of observations (rows) of a set of variables (columns). E.g. book titles and prices.

- Similar to a matrix but variables in different columns can have different types.

- Always the same number of entries in each row, although some may be missing (`NA`).

- Can be formed by reading in data e.g. from a spreadsheet, or constructed using the function `data.frame`.

Chick weights data frame

```
   weight      feed
1     179 horsebean
2     160 horsebean
3     136 horsebean
4     227 horsebean
.
.
.
```

Lists A data frame is a kind of list, which is a vector of objects of possibly different types. For example, an employee record might be created by

```
Empl <- list(employee = "Anna", spouse = "Fred", children
= 3, child.ages = c(4, 7, 9))
```

Components are always numbered and may be referred to by number. e.g. `Empl[[2]]`. If they are named, can also be referred to by name using the $ operator eg. `Empl$spouse`

Note that `Empl[4]` is a list of length 1, while `Empl[[4]]` is a numeric vector of length 3.

Keeping track of objects

- Once you have created some objects, how do you remind yourself what you called them?
- Use a function such as `ls()` or `str()`.
- `ls()` lists the names of all the objects in your workspace.
- `str(Empl)` gives a little information about the object `Empl`.

```
str(Empl)
 List of 4
 $ employee    :  chr "Anna"
 $ spouse      :  chr "Fred"
 $ children    :  num 3
 $ child.ages  :  num [1:3] 4 7 9
```

More functions `paste`

- `paste` adds together character vectors: `paste(c(1, 2), c('x', 'y', 'z'))` is a character vector of length 3

  ```
  [1] "1 x" "2 y" "1 z"
  ```

  Notice the recycling of the first argument.

- The pieces of the result can be joined together using the argument `collapse`:

  `paste(c(1, 2), c('x', 'y', 'z'), collapse=' ')` is a character vector of length 1,

  ```
  "1 x 2 y 1 z"
  ```

Yet more functions `sort,table`

- `sort(x)` sorts the vector `x` into increasing order

- `sort(x, decreasing=TRUE)` sorts the vector `x` into decreasing order

- `table(x)` creates a table showing the count of elements equal to each value. Most useful where `x` is a vector of factors or integers

- e.g. `table(rpois(20, 5))` gives

  ```
  2 3 4 5 6 7 8
  3 2 2 6 3 3 1
  ```

Installing R on your own machine

- Visit CRAN at **`http://cran.r-project.org/`**

- Find the appropriate binary
  - Windows: **`http://cran.r-project.org/bin/windows/base/release.htm`**
  - Mac: select as appropriate
  - Linux: select as appropriate.

- After installation, create a shortcut which starts in your folder (using right-click/properties)

- Update packages via the packages menu or using **`update.packages()`**

Books

In addition to the extensive documentation and help system that is included in R there are three main books that we recommend.

‘A First Course in Statistical Programming with R’ by W. John Braun and Duncan Murdoch, ISBN 0-521-69424-8

‘Introductory Statistics with R’ by Peter Dalgaard, ISBN 0-387-95475-9

- a very good introduction to R that includes many biostatistical examples and covers most of the basic statistics covered in this course.

‘Modern Applied Statistics with S’ by Bill Venables and Brian Ripley, ISBN 0- 387-95457-0

- a comprehensive text that details the S-PLUS and R implementation of many statistical methods using real datasets.

Practical

- Practical is a mixture of getting you to enter commands and solving problems
- If you use the same computer each week the files from your last session will be available. So note down the number of the computer you have used.
- Use a script and save the script to the H: drive. Save it with a file name that includes your name.
- May want to bring a memory stick to make a copy of your script or email the script to yourself.
- Web browsing, facebook etc are banned from practical sessions.
- To quit R type `q()`. Then select No when asked if you want to save the session.

Transformation Methods (continued)

See you on Monday

Our next meeting is in the lecture theatre in the Statistics Department SPR1.

The first problem sheet is online at

`http://www.stats.ox.ac.uk/~nicholls/PartASSP/`

and due Monday morning at 9am (see tray by mailboxes in SPR1).

Homework: install R and complete the practical in your own time.