
A beginners guide to solving biological problems in R

Robert Stojnić (rs550), Laurent Gatto (lg390),
Rob Foy (raf51) and John Davey (jd626)

Course material:
<http://logic.sysbiol.cam.ac.uk/teaching/Rcourse/>

Original slides by Ian Roberts and Robert Stojnić

Day 1 schedule

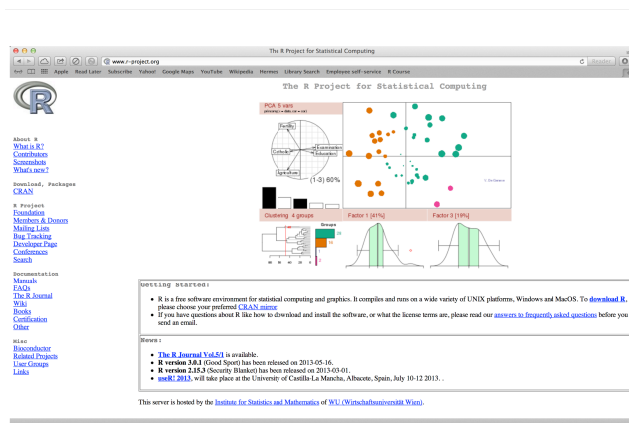
1. Introduction to R and its environment
2. Data structures
3. Data analysis example
4. Programming techniques
5. Statistics

Introduction to R and its environment

1

What's R?

- A statistical programming environment
 - based on S
 - Suited to high level data analysis
- Open source & cross platform
- Extensive graphics capabilities
- Diverse range of add-on packages
- Active community of developers
- Thorough documentation



www.r-project.org

Various platforms supported

- Release 3.0.1 (May 2013)
 - Base package
 - Contributed packages (general purposes extras)
 - ~4700 available packages
- Download from <http://www.stats.bris.ac.uk/R/>
- Windows, Mac and Linux versions available
- Executed using command line, or a graphical user interface (GUI)
- On this course, we use the RStudio GUI (www.rstudio.com)
- Everything you need is installed on the training machines
- If you are using your own machine, download both R and RStudio

Getting Started

- R is a program which, once installed on your system, can be launched and is immediately ready to take input directly from the user
- There are two ways to launch R:
 - 1) From the command line (particularly useful if you're quite familiar with Linux)
 - 2) As an application called RStudio (very good for beginners)

Prepare to launch R

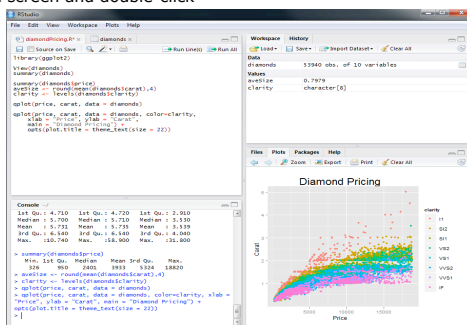
From command line

- To start R in Linux we need to enter the Linux console (also called Linux terminal and Linux shell)
- To start R, at the prompt simply type:
\$ R
- If R doesn't print the welcome message, call us to help!

Prepare to launch R

Using RStudio

- To launch RStudio, find the RStudio icon in the menu bar on the left of the screen and double-click



The Working Directory (wd)

- Like many programs R has a concept of a working directory (wd)
- It is the place where R will look for files to execute and where it will save files, by default
- For this course we need to set the working directory to the location of the course scripts
- At the command prompt in the terminal or in RStudio console type:

```
> setwd("R_course/Day_1_scripts")
```

- Alternatively in RStudio use the mouse and browse to the directory location
- Tools » Set Working Directory » Choose Directory...

Basic concepts in R command line calculation

- The command line can be used as a calculator. Type:

```
> 2 + 2  
[1] 4
```

```
> 20/5 - sqrt(25) + 3^2  
[1] 8
```

```
> sin(pi/2)  
[1] 1
```

- Note: The number in the square brackets is an indicator of the position in the output. In this case the output is a 'vector' of length 1 (i.e. a single number). More on vectors coming up...

Basic concepts in R variables

- A variable is a letter or word which takes (or contains) a value. We use the assignment 'operator', `<-`

```
> x <- 10  
> x  
[1] 10  
> myNumber <- 25  
> myNumber  
[1] 25
```

- We can perform arithmetic on variables:

```
> sqrt(myNumber)  
[1] 5
```

- We can add variables together:

```
> x + myNumber  
[1] 35
```

Basic concepts in R variables

- We can change the value of an existing variable:

```
> x <- 21
> x
[1] 21
```

- We can set one variable to equal the value of another variable:

```
> x <- myNumber
> x
[1] 25
```

- We can modify the contents of a variable:

```
> myNumber <- myNumber + sqrt(16)
[1] 29
```

Basic concepts in R functions

- **Functions** in R perform operations on **arguments** (the input(s) to the function). We have already used **sin(x)** which returns the sine of **x**. In this case the function has one argument, **x**. Arguments are *always* contained in parentheses, i.e. curved brackets **()**, separated by commas.

- Try these:

```
> sum(3, 4, 5, 6)
[1] 18
> max(3, 4, 5, 6)
[1] 6
> min(3, 4, 5, 6)
[1] 3
```

- Arguments can be named or unnamed, but if they are unnamed they must be ordered (we will see later how to find the right order).

```
> seq(from=2, to=10, by=2)
[1] 2 4 6 8 10
> seq(2, 10, 2)
[1] 2 4 6 8 10
```

Basic concepts in R vectors

- The basic data structure in R is a **vector** – an ordered collection of values. R even treats single values as 1-element vectors. The function **c()** *combines* its arguments into a vector:

```
> x <- c(3, 4, 5, 6)
> x
[1] 3 4 5 6
```

- As mentioned, the square brackets **[]** indicate position within the vector (the **index**). We can extract individual elements by using the **[]** notation:

```
> x[1]
[1] 3
> x[4]
[1] 6
```

- We can even put a vector inside the square brackets (vector indexing):

```
> y <- c(2, 3)
> x[y]
[1] 4 5
```

Basic concepts in R vectors

- There are a number of shortcuts to create a vector. Instead of:

```
> x <- c(3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
```

- we can write:

```
> x <- 3:12
```

- or we can use the **seq()** function, which returns a vector:

```
> x <- seq(2, 10, 2)
```

```
> x
```

```
[1] 2 4 6 8 10
```

```
> x <- seq(2, 10, length.out = 7)
```

- > x

```
[1] 2.00000 3.33333 4.66667 6.00000 7.33333 8.66667 10.00000
```

- or the **rep()** function:

```
> y <- rep(3, 5)
```

- > y

```
[1] 3 3 3 3 3
```

```
> y <- rep(1:3, 5)
```

```
> y
```

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```

Basic concepts in R vectors

- We have seen some ways of extracting elements of a vector. We can use these shortcuts to make things easier (or more complex!)

```
> x <- 3:12
```

```
> x[3:7]
```

```
[1] 5 6 7 8 9
```

```
> x[seq(2, 6, 2)]
```

```
[1] 4 6 8
```

```
> x[rep(3, 2)]
```

```
[1] 5 5
```

- We can add an element to a vector

```
> y <- c(x, 1)
```

```
> y
```

```
[1] 3 4 5 6 7 8 9 10 11 12 1
```

- We can glue vectors together

```
> z <- c(x, y)
```

```
> z
```

```
[1] 3 4 5 6 7 8 9 10 11 12 3 4 5 6 7 8 9 10 11 12 1
```

Basic concepts in R vectors

- We can remove element(s) from a vector

```
> x <- 3:12
```

```
> x[-3]
```

```
[1] 3 4 6 7 8 9 10 11 12
```

```
> x[-(5:7)]
```

```
[1] 3 4 5 6 10 11 12
```

```
> x[-seq(2, 6, 2)]
```

```
[1] 3 5 7 9 10 11 12
```

- Finally, we can modify the contents of a vector

```
> x[6] <- 4
```

```
> x
```

```
[1] 3 4 5 6 7 4 9 10 11 12
```

```
> x[3:5] <- 1
```

```
> x
```

```
[1] 3 4 1 1 1 4 9 10 11 12
```

- Remember! **Square** brackets for indexing **[]**, **parentheses** for function arguments **()**.

Basic concepts in R

vector arithmetic

- When applying all standard arithmetic operations to vectors, application is element-wise

```
> x <- 1:10
> y <- x*2
> y
[1] 2 4 6 8 10 12 14 16 18 20
> z <- x^2
> z
[1] 1 4 9 16 25 36 49 64 81 100
```

- Adding two vectors

```
> y + z
[1] 3 8 15 24 35 48 63 80 99 120
```

- Vectors don't have to be the same length (what's this?)...

```
> x + 1:2
[1] 2 4 4 6 6 8 8 10 10 12
```

- but that doesn't always work

```
> x + 1:3  (...?)
```

Writing scripts with Rstudio

Typing lots of commands directly to R can be tedious. A better way is to write the commands to a file and then load it into R.

- Click on **File -> New** in Rstudio
- Type in some R code, e.g.

```
x <- 2 + 2
print(x)
```

- Click on **Run** to execute the **current line**, and **Source** to execute the **whole script**



Sourcing can also be performed manually with `source("myScript.R")`

Getting Help

- To get help on any R function, type `?` followed by the function name. For example:

```
> ?seq
```
- This retrieves the syntax and arguments for the function. You can see the default order of arguments here. The help page also tells you which **package** it belongs to.
- There will typically be example usage, which you can test using the **example** function:

```
> example(seq)
```
- If you can't remember the exact name type `??` followed by your guess. R will return a list of possibilities

```
> ??rint
```

Interacting with the R console

- R console symbols
 - `;` end of line
 - Enables multiple commands to be placed on one line of text
 - `#` comment
 - indicates text is a comment and not executed
 - `+` command line wrap
 - R is waiting for you to complete an expression
- **Ctrl-c** or **escape** to clear input line and try again
- **Ctrl-l** to clear window
- Press **q** to leave help (using R from the terminal)
- Use the **TAB key** for command auto completion
- Use **up and down arrows** to scroll through the command history

R packages

- R comes ready loaded with various libraries of functions called **packages**. e.g. the function **sum()** is in the **base** package and **sd()**, which calculates the standard deviation of a vector, is in the **stats** package
- There are 1000s of additional packages provided by third parties, and the packages can be found in numerous server locations on the web called **repositories**
- The two repositories you will come across the most are
 - **The Comprehensive R Archive Network (CRAN)**
 - **Bioconductor**
- CRAN is mirrored in many locations. Set your local mirror in RStudio using Tools -> Options, and choose a CRAN mirror
- Set the Bioconductor package download tool by typing:

```
> source("http://bioconductor.org/biocLite.R")
```
- Bioconductor packages are then loaded with the `biocLite()` function:

```
> biocLite("PackageName")
```

R packages

- 4700+ packages on CRAN:
 - Use CRAN search to find functionality you need:
<http://cran.r-project.org/search.html>
 - Or, look for packages by theme:
<http://cran.r-project.org/web/views/>
- 670+ packages in Bioconductor:
 - Specialised in genomics:
<http://www.bioconductor.org/packages/release/bioc/>
- **Other repositories:**
- 1600+ projects on R-forge:
 - <http://r-forge.r-project.org/>
- R graphical manual:
 - <http://rgm3.lab.nig.ac.jp/RGM>

Bottomline: **always** first look if there is already an R package that does what you want before trying to implement it yourself

Exercise: Install Packages Matrix and aCGH

- Matrix is a CRAN extras package
 - Use `install.packages()` function...
`install.packages("Matrix")`
 - or in RStudio goto Tools → Install Packages... and type the package name
- aCGH is a BioConductor package (www.bioconductor.org)
 - Use `biocLite()` function
`biocLite("aCGH")`
- R needs to be told to use the new functions from the installed packages
 - Use `library(...)` function to load the newly installed features
`library("Matrix") # loads matrix functions`
`library("aCGH") # loads aCGH functions`
 - `library()`
 - Lists all the packages you've got installed locally
