Introduction to R and Bioconductor: Computer Lab

Bjørn-Helge Mevik (b.h.mevik@usit.uio.no), Research Computing Services, USIT, UiO

(based on original by Antonio Mora, biotek)

Exercise 1. Fundamentals of R

R has been defined as "a language and environment for statistical computing and graphics" [1]. R is free and open-source and it has been extended through packages. If you don't have it installed in your computer, you can download it.

- For Windows machines, go to http://cran.ii.uib.no/bin/windows/base/, download the file and follow the instructions.
- For Linux machines, go to http://cran.r-project.org/src/base/R-2/ and get R's source code (R-2.14.0.tar.gz); decompress it and install it (use: "configure --prefix=/your-path/R --with-tcltk --without-cairo --with-libpng –with-jpeglib", then "make" and "make check"). *Note that R is in the package repository of many Linux distributions, so check with your yum or apt-get!*
  (*Notice that /your-path/ specifies the path from the root directory to the R directory. In Windows can be something like: C:/Program Files/R/ In Linux, something like /usr/local/R/*)

A tutorial can be found typing "help.start()". However, we will review the main features:

- Start and quit: You can start R in Windows just like any other Windows application; to start R in Linux, type "R" in a terminal (assuming that R is already on your path); the R prompt will appear. To quit, type "q()".

- Ordinary arithmetic operators: After the R prompt, type some arithmetic calculations (using +, -, *, / and ^, grouped with parenthesis). See the way in which R displays the results. Try:
  > sqrt(2)

> sin(pi)
> exp(1)
> log(10)
> log(10, base = 10)
Try writing two arithmetic operations separated by a semicolon; see how R shows both results.

- Assign value to a variable: Use “<-”; for instance:
> a <- 5
> a <- 5 + 2
Now, type “a”. Now, try a + 2. What happens if you try a + b?

- List variables: To list all current variables in the workspace, type: “ls()”. Which variables are present in your workspace so far?

- Help: “?” is useful to access the help. For instance, “?mean” shows information about the “mean” function; ?rep describe a function to create a vector of repeated elements. Use “? function-name” each time you don't understand the goal of a function. Read the help using the space bar and quit help using “q”.

- Use the up and down arrow keys to access the command history.

- Creating a vector: Vectors have elements of the same type. For instance:
> vec <- c(2, 3, 4)
The first element of the vector is considered to be at position 1. Therefore, you can check the second element of this vector using vec[2]. Create two vectors “a” and “b” with at least three elements each and join them using d <- c(a, b). Try:
> d[1:3]
> d[d < 4]
What do you observe?. Try vec[-1] and you should get everything but the first element.

- Creating a list: Lists can have elements of different types. For instance:
> l <- list(1, 2, "hi")
Check the third element of the list using l[[3]]. We can also create a named (hash-like) list:
> m <- list(a = 1, b = 2, d = "hi")
Now, you can extract the value for “b” using the “$” sign:
> m$b

- Printing: Use print() or just type the name of the variable. Try printing a and m.
  > print(a)
  > print(m)

- 'If' and 'For': Try the following examples:
  if (1 > 0) print("hello")
  for (i in 1:5) { print(i) }
  for (i in list("a", "b", TRUE)) { print(i) }
  What are these commands doing?

- Creating a matrix: Try:
  > mat <- matrix(1:9, nrow = 3, ncol = 3)
  How does the matrix look like?
  Now try:
  > for (i in 1:3) { print(mat[i, 3]) }
  What happened here?
  Other ways to get the same result are: mat <- cbind(c(1,2,3), c(4,5,6), c(7,8,9)), or: mat <- rbind(c(1,4,7), c(2,5,8), c(3,6,9)), where “cbind” stands for “column bind” and “rbind” stands for “row bind”.
  We can give names to a matrix using:
  > rownames(mat) <- c("Obs1", "Obs2", "Obs3")
  > colnames(mat) <- c("Var1", "Var2", "Var3")

- Data Frames: Data frames contain several variables (columns) and several experimental units (rows). Try:
  > d1 <- data.frame(x = 1, y = 1:10)
  Check d1. What does d1$y give you?

- Plot: You can plot using a variety of options. Try, for instance:
  > plot(x = 1:10)
  > plot(x = 1:10, y = (1:10)^2)
  > plot(x = 1:10, y = (1:10)^2, type = "p", col = "red", main = "My Graph", pch="*")
  Now try the last exercise with type = "l".
  Use “demo(graphics)” to see many plotting options.

- dev.new(): Try the following plot:
  > for (i in 1:5) { plot(i:10) }
  What happens? Now, add the “dev.new()” call:
  > for (i in 1:5) {dev.new(); plot(i:10) }

What difference can you observe? What is dev.new() doing?

- Save and Load workspace: It is possible to save all variables in the workspace to use them later. Use:
  > save(a, vec, mat, file = "variabs.RData")
  Quit the R program as explained before (specify that you don't want to save the workspace) and re-enter. Check that there are no variables using ls() and now load the variables that you just saved using:
  > load("variabs.RData")
  Check that “a”, “vec” and “mat” are now in your workspace.

- Read and write text: The working directory (i.e., the directory where the files you produce will be stored) can be get or specified using “getwd()” and “setwd()”, respectively. Find out where is your working directory.
  Now, we will create a text file containing the matrix that we called “mat”:
  > write.table(mat, "mytable.txt")
  Check that a file called “mytable.txt” was generated at the working directory. In order to read this file, we can use the read.table() function:
  > c <- read.table("mytable.txt")
  Check “c”, rownames(c) and colnames(c).
  Several options to read and write are included in the documentation; f.ex., append=TRUE will add the information to an existing file.
  Now you will open the table using the R editor:
  > cnew <- edit(c)
  Change some values at the editor and press “Quit”. Now check your changes in “cnew”.
  Try:
  > cnew$Var1
  > cnew[, 2]
  > cnew[3,]
  > c[c$Var1== "3",]
  What happened in each case?

- Creating a function: Use: function(args) { exprs }. For instance:
  > suma <- function(a, b){ a + b }
  Now, try:
  > suma(1, 2)
  Another example:
  > suma2 <- function (a, b) { print(a); print(b); a + b }
  Check suma2(2, 3)

- Scripts: You don't have to type all your commands in the command line every time you need it. For repetitive tasks, you can create scripts using a text editor such as notepad: Copy some of the exercises you have already done to a text file (do not include the ">" symbol) and save it at your working directory as "name-of-the-script.R". Then run it using: source(file = "name-of-the-script.R").

Some words on packages:

- Packages: Functions belong to packages. The base R distribution comes with a library of packages, and it is possible to install other packages in the library. If you want to use a function from a certain package, the package must be already installed and loaded. Check which packages are installed in you library with the command library().
- Bioconductor is a group of packages for bioinformatics. Check the available packages at: http://www.bioconductor.org/packages/release/bioc/.
  After installation, you can load any of the installed packages from the library in order to add their functionalities, using: library(name-of-the-package).

- Some packages also include a "data" directory, containing data to be used as an example. Data sets are stored as "data frames". data() will list all available data sets in your loaded packages; data(package="GO") list data sets for the GO package; data("CO2") loads the data set CO2, while names(CO2) gives us all variables in the data set (after it has been loaded); we can check each variable using "data-set$variable-name", as in CO2$Type.

Optional:

Regular expressions:

Try the following code and find out what is doing:

```
> grep("[a-z]", letters)
>  txt  <-  c("arm",  "foot",  "lefroo",  "bafoobar",
"gorilla", "armadillo", "far", "tate")
> i <- grep("foo",txt)
> txt[i]

> i=grep("^l.*",txt)
> txt[i]
> i=grep("^f[oa]",txt)
> txt[i]
> i=grep("^[fg][oa]",txt)
> txt[i]
```

```
> i=grep(".*o$",txt)
> txt[i]
> i=grep("^...o{2}b..$",txt)
> txt[i]

> txt <- c("The", "licenses", "for", "most", "software",
"are",   "designed",   "to",   "take",   "away",   "your",
"freedom",  "to",  "share",  "and",  "change",  "it.",  "",
"By",  "contrast,",  "the",  "GNU",  "General",  "Public",
"License",  "is",  "intended",  "to",  "guarantee",  "your",
"freedom",   "to",   "share",   "and",   "change",   "free",
"software",    "--",    "to",    "make",    "sure",    "the",
"software", "is", "free", "for", "all", "its", "users")
> ot <- sub("[b-e]",".", txt)
> ot2 <- sub("[b-e]{2}","..", txt)
```

Exercise 2. Statistics using R

Here we will give a very short introduction to some basic statistics concepts. For a more in-depth study, check any statistics book including R code. First, we will use basic descriptive statistics with R. Type:

```
> x <- 1:20
```

Sums:

$$\sum_n x = x_1 + x_2 + ... + x_n$$

```
> sum(x)
```

Mean:

$$\bar{x} = \frac{1}{n} \sum_n x$$

```
> mean(x)
```

Cumulative sum:

$$\bar{x}_i = \sum_{j=1,i} x_j$$

```
> cumsum(x)
```

Range:

Minimum and maximum value

```
> range(x)
```

Variance:

$$\sigma(x)=\frac{(x_1-\bar{x})^2+(x_2-\bar{x})^2+...+(x_n-\bar{x})^2}{n-1}$$

```
> var(x)
```

The standard deviation is the square root of the variance:

$s=\sqrt{\sigma}$

```
> sd(x)
> sqrt(var(x))
```

To get the number of times of each experimental datum:

```
> table(x)
```

A bar plot could be useful to visualize data. Try:

```
> barplot(x, xlab="observation", ylab="value")
> dev.new()
> barplot(table(x), xlab="value", ylab="frequency")
```

R is also popular for probability theory. Here, we will try some simple examples.

In order to randomly select one element of a vector, where each element has a certain probability “p” of being chosen, we use the “sample” function. For example, to roll a fair die, try:

```
> sample(1:6, size = 1)
```

You will get numbers from 1 to 6 with the same probability each time.

If each element has a different probability, as in a loaded die, try:

```
> k <- 1:6
> p <- c(0.5,0.1,0.1,0.1,0.1,0.1)
> sample(k, size = 1, prob = p)
```

Here, it is more probable to get the number 1.

Let's toss a coin 10 times. Say that heads=1 and tails=0, and, given that these options are kept during the whole process, it is said that there is “replacement”, so:

```
> sample(0:1, size = 10, replace = TRUE)
```

(If replacement is set to FALSE, it means that, after the first trial (0 or 1), we just have one option left (1 or 0) and, then, the population will disappear, i.e., the size can't be bigger than 2).

Write the code for choosing 30 cards from a deck, given that you remove the card after choosing it (replace=FALSE) and given that you return the card each time (replace=TRUE).

References

[1] R website, http://www.r-project.org/,
[2] VERZANI, J., Using R for Introductory Statistics, Chapman & Hall / CRC, 2005