Brief course in R.

Sebastian E. Ramos-Onsins.
Centre for Research in Agricultural Genomics (CRAG)

November 11^{th}, 2013

Contents

1 Basic Concepts to Start with R

R is an application used to perform a large number of statistical, mathematical, graphics and other operations that are useful for statistical analysis. The application shows an environment where the user can introduce commands (yes, it's a command line-based application), functions and libraries of functions. The user can also introduce data easily and obtain high quality output plots.
R is a free software. You can download it from http://www.r-project.org. There is a large number of additional functions that can be mainly downloaded from http://cran.r-project.org.
R has an extended help included in the program, although it is not so explicative for beginners. A number of manuals are available for people interested. For example: ”Introductory Statistics with R” (Dalgaard, 2002) or ”A Handbook of Statistical Analysis Using R” (Everitt and Hothorn, 2006).
When you open R, a command-line window appears. You can write directly any command and the result will be output after pressing ”return”. You can also open a editor of R scripts to keep all the commands you write (see the Menu). This option is very useful for making functions and save it in a file with ”.r” extension. Then, these functions (or a number of commands in a R file) can be used when you need. You simply have to open the file and copy all what you want inside R command-line window.

1.1 Basic commands and operations

1.1.1 Help

The command **help(function)** and **apropos(”*any_name*”)** are basic functions that allow the user to find and use correctly the commands or functions in R. Using the command **help.start()** you can get access to the complete documentation help menu.

```
> apropos("norm")
[1] "dlnorm" "dnorm" "norm"
[4] "normalise" "normalize_spectrum" "normalizePath"
[7] "plnorm" "pnorm" "qlnorm"
[10] "qnorm" "qqnorm" "qqnorm.default"
[13] "rlnorm" "rnorm" "spnorm"
[16] "sum_spectra_norm"
> help(rnorm)
```

1.1.2 R as a calculator

Using R as a calculator. Sum, divide, square root, power...

```
> 3 + 45
[1] 48
> 48 * 1.2
[1] 57.6
> sqrt(49)
[1] 7
> 8^34
[1] 5.070602e+30
> log(23*exp(4))
[1] 7.135494
```

1.1.3 Keeping the data

It is possible to keep values in variables, not only scalars but also vectors, matrices, or even lists of values.

(i) Keep values in a variables: Example:

```
> x <- 45
> y <- 3.4
> z <- x^2 + 2*y - x/y
> z
[1] 2018.565
```

(ii) Keep data in vectors, matrices or arrays: Using the command **c(*value,values,...*)** is possible to keep values in a vector. Example:

```
> f <- c(1:4)
> f
[1] 1 2 3 4
> g <- c(3,5,6)
> g
[1] 3 5 6
> h <- c(f,g)
> h
[1] 1 2 3 4 3 5 6
```

Exercise 1.1.3.ii: Do an arithmetical sum of two vectors

Using the form **matrix(*values*,nrow=*dim_row*,ncol=*dim_col*)** or the generic command **array(*values*,dim=c(*size_dim1*,*size_dim2*,*size_dim3*,*...*))** it is possible to work with large matrices of values. A matrix must have a unique type of variable, that is, the values are numerical or categorical, but not both at the same time (if you include a letter in a matrix with numbers, all the numbers will be treated as characters).
Example:

```
> m <- matrix(,nrow=4,ncol=3)
> m
  [,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
[3,] NA NA NA
[4,] NA NA NA
> m <- matrix(0,nrow=4,ncol=3)
> m
  [,1] [,2] [,3]
[1,] 0 0 0
[2,] 0 0 0
[3,] 0 0 0
[4,] 0 0 0
> m <- matrix(c(1:12),nrow=4,ncol=3)
> m
  [,1] [,2] [,3]
[1,] 1 5 9
```

```
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
> m[2,3]
[1] 10
> m <- matrix(c(1:12),nrow=4,ncol=3,byrow=TRUE)
> m
  [,1] [,2] [,3]
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
[4,] 10 11 12
> m[2,3] #show the value in row 2, column 3
[1] 6
> r <- array(c(1:12),dim=c(2,3,2))
> r
, , 1

  [,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

, , 2

  [,1] [,2] [,3]
[1,] 7 9 11
[2,] 8 10 12

> r[2,3,2]
[1] 12
> r[2,,1]
[1] 2 4 6
> r[2,-2,1]
[1] 2 6
```

(iii) The data.frame and the list. The data frame is a set of values (numerical or categorical) that are included in a unique frame and are grouped in a matrix frame. The main difference with a matrix is the possibility to work with both (numerical and categorical) in the same variable. A data frame must have the same number of rows for each defined column. The command is **data.frame(*name_var*=*values*,...)**.
Example:

```
> d <- data.frame(x=0,y=1:5,cat=c("type1","type2","type1","type1","type2"))
> d
  x y cat
1 0 1 type1
2 0 2 type2
3 0 3 type1
4 0 4 type1
5 0 5 type2
> d$y
[1] 1 2 3 4 5
> d$cat
[1] type1 type2 type1 type1 type2
```

```
Levels:  type1 type2
> d$x[1]
[1] 0
> d$y
[1] 1 2 3 4 5
> d$y>3
[1] FALSE FALSE FALSE TRUE TRUE
> d$cat[3]
[1] type1
Levels:  type1 type2
> d[2,]
  x y cat
2 0 2 S
```

Exercise 1.1.3.iii: Select those values of "cat" that have a value of "y" smaller than 4.

A **list** is any group of variables that have no restriction. Any kind of varible can be included, no matter if the number of rows are different or the variables are categorical or numerical. It is useful for packaging a number of values and variables with a unique name. Example:

```
> l <- list(r=r,f=f,d=d)
> l
$r
, , 1

   [,1][,2][,3]
[1,] 1 3 5
[2,] 2 4 6

, , 2

   [,1][,2][,3]
[1,] 7 9 11
[2,] 8 10 12

$f
[1] 1 2 3 4
$d
  x y cat
1 0 1 A
2 0 2 S
3 0 3 N
4 0 4 Q
5 0 5 E
```

Finally, you can use the function **class(*variable_name*)** to know what is the the type of the variable you are interested in. Example:

```
> f <- c(1:10)
> g <- f*pi
> class(f)
[1] "integer"
> class(g)
```

```
[1] "numeric"
> h <- c(f,g)
> class(h)
[1] "matrix"
```

1.2 Interesting functions in R

There is a large number of functions in R. Here I only introduce you those functions that I consider useful and interesting for this short practical course.

1.2.1 Commands for general purposes

getwd() setwd() library() source() read.table() fread() write.table() which() cbind() rbind() length() sprintf()

(i) getwd(): Shows the working directory. All files you load or save will be in this directory if it is not specified. Example:

```
> getwd()
```

(ii) setwd(): Changes the working directory to the one you specify. Example:

```
> f <- "/Users/sebas/Desktop/PracticalCoal"
> setwd(f)
```

(iii) library(): This command is necessary to load packages of functions. Example:

```
> library(datasets)
```

(iv) source(): This is useful to load specific command from a file.Example:

```
> source("./practica_funciones.R")
```

(v) write.table(): This is useful to save data or results to a file.

```
Usage:
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n",
na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape",
"double"))
```

x is the variable you want to write in the file. *file* is the name of the file, between quotes. With *append* is possible to add more data to the file without erasing the content (T) or not (F). *quote* put or not quotes to the values of the variable. *sep* indicates the spacing character. *eol* indicates the character to include at end of the row in a table. *na* is useful in case Non Available data is included. *row.names* indicates if you want the name of the rows are written in the file.
Example:

```
> library(datasets)
> cars[1 : 5, ]
  speed dist
1 4 2
2 4 10
3 7 4
```

```
4 7 22
5 8 16
> write.table(x=cars,file="data_example.txt",append=F,
quote=F,eol="\n",dec=".",row.names=F)
```

(vi) read.table(): Include data from a file.

```
Usage:
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".", row.names,
col.names, as.is = !stringsAsFactors, na.strings = "NA", colClasses = NA, nrows
= -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white =
FALSE, blank.lines.skip = TRUE, comment.char = "#", allowEscapes = FALSE, flush
= FALSE, stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
encoding = "unknown")
```

Example:

```
> data.cars <- read.table(file="data_example.txt",header=T)
> data.cars[1 : 5, ]
  speed dist
1 4 2
2 4 10
3 7 4
4 7 22
5 8 16
```

(vii) fread(): Read data from a file much faster than read.table. You have to install the library ”data.read” to use it.

(viii) which(): Indicates the position of the value that you are looking for. Example:

```
> data.cars$dist > 50
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE
[25] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE FALSE
[37] FALSE TRUE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[49] TRUE TRUE
> which(data.cars$dist > 50)
[1] 22 23 26 33 34 35 38 41 42 43 44 45 46 47 48 49 50
```

(ix) cbind(): Merge the columns of two or more variables in the order specified. Example:

```
> g <- data.cars$speed
> speed2 <- cbind(g,2*g)
> speed2[1 : 5, ]
g
[1, ] 4 8
[2, ] 4 8
[3, ] 7 14
[4, ] 7 14
[5, ] 8 16
```

(x) rbind(): Merge the rows of two or more variables in the order specified.

(xi) length(): Gives the size of a vector. Example:

```
> g <- data.cars$speed
> length(g)
[1,] 50
```

(xii) sprintf(): shows text that can include the values of other variables. Example:

```
> a <- c(1.4,3.4)
> t <- c("variable","answer")
> w <- sprintf("The name of the %s is %f, and the %s is %f",t[1],a[1],t[2],a[2])
> w
```

1.2.2 Some algorithmic, statistical and plotting commands

mean() var() apply() quantile() sample() runif() qexp() plot() hist() pdf()

(i) mean(), var(), sd(): Calculate the mean (var or standard deviation, sd) for the values you include. In case of var, when several columns are included, it also calculates de covariance. An interesting option is the possibility to include the "NA" values (na.rm=F) or exclude from the analysis (na.rm=T). Example:

```
> mean(data.cars$dist,na.rm=T)
[1] 42.98
> var(data.cars$dist,na.rm=T)
[1] 664.0608
> var(data.cars,na.rm=T)
         speed      dist
speed 27.95918 109.9469
dist 109.94694 664.0608
```

(ii) apply(): This command allows to use a function over the rows, the columns or both in a matrix.

```
Usage:
apply(X, MARGIN, FUN, ...)
Arguments
X is an array, including a matrix.
MARGIN is a vector giving the subscripts which the function will be applied
over.  1 indicates rows, 2 indicates columns, c(1,2) indicates rows and columns.
FUN the function to be applied (e.g., sum, var ...)
...  optional arguments to FUN.
```

Example:

```
> apply(data.cars,2,mean)
speed dist
15.40 42.98
```

(iii) sample(): This function pick a number of elements from a vector, with or without replacement. Useful for bootstrap resampling. Example:

```
> # 100 Bernoulli trials
> b <- sample(c(0,1), 100, replace = TRUE)
> mean(b)
[1] 0.49 > var(b)
[1] 0.2524242
```

(iv) runif(), rpois(), rnorm(), rexp(): Give a number of values following a Uniform (Poisson, or other) distribution. Example:

```
> u <- runif(n=1000,min=-2,max=2)
> p <- rpois(n=1000,lambda=10)
> n <- rnorm(n=1000,mean=0,sd=1)
> apply(cbind(u,p,n),2,mean)
      u      p      n
0.04939483 10.07100000 0.01193145
> apply(cbind(u,p,n),2,var)
      u      p      n
1.317368 9.373332 0.948162
```

(v) qpois(), qnorm(), qexp(): Starting from the density distribution, we pick a value from the distribution when the *P*-value is provided. Example:

```
> t_rate <- 1/1000
> uu <- runif(1)
> uu
[1] 0.7117541
> t1 <- qexp(p=uu,rate=t_rate)
> t1
[1] 1242.9414
```

(vi) plot: Plot elements of variables within a graphical display in MANY different ways. Here we only show a simple 2D plot of one variable versus another. Example:

```
> x <- rnorm(1000)
> y <- 2*x*x + rnorm(1000)
> plot(x,y,xlab="x=Normal",ylab="y=2*x^2+Normal")
```

(vi) pdf: You can send the results to a pdf file. All the plot ouputs will be sent to the defined pdf. the file is finally created with the command **dev.off()**:

```
> pdf(file="Example_plot.pdf",height=6,width=10)
> plot(x,y,xlab="x=Normal",ylab="y=2*x^2+Normal")
> dev.off()
```

(vii) quantile(): Calculate the quantiles for the distribution you include. A vector of probabilities can be specified. Example:

```
> n <- rnorm(1000)
> quantile(n)
      0%       25%       50%       75%       100%
-2.96587439 -0.64760592 0.02209920 0.64540246 2.93180125
> quantile(n,probs=c(0.025,0.975)
      2.5%       97.5%
-1.954497 1.887044
```

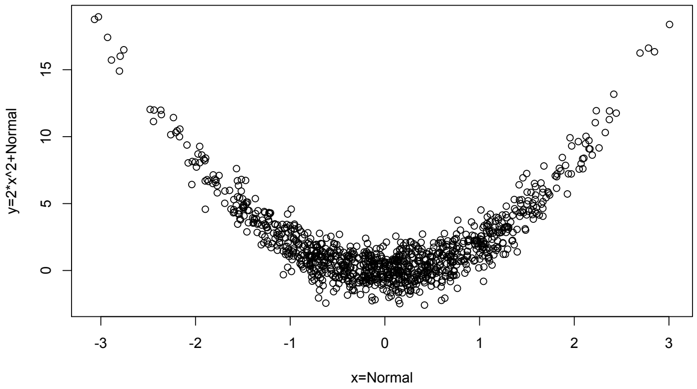

Figure 1: Example of the command plot.

(viii) hist(): Shows the histogram for a vector. Only few options are shown. *n* is the data, *breaks* are the number of segments to divide the data, *freq* shows the absolute number of values in the y-axis (T) or instead the proportion (F). Example:

```
> u <- runif(n=10000,min=-2,max=2)
> p <- rpois(n=10000,lambda=10)
> n <- rnorm(n=10000,mean=0,sd=1)
> par(mfrow=c(1,3))
> hist(u,breaks=30,main="Uniform.  min=-2, max=2",freq=F)
> hist(p,breaks=30,main="Poisson.  lambda=01",freq=F)
> hist(n,breaks=30,main="Normal",freq=F)
```

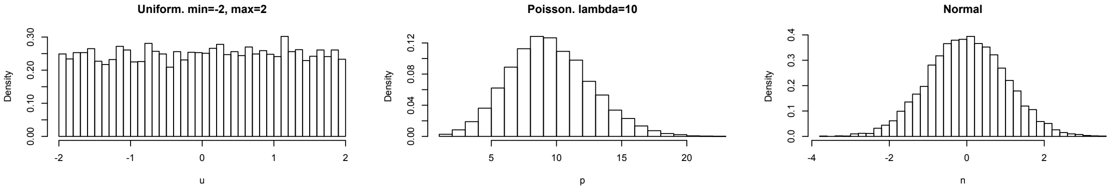

Figure 2: Example of the command hist.

1.2.3 A bit about programming commands

function() for() while() if()

(i) function(): This command is useful to build a list of commands or orders that are interesting to run together (and usually many times). The function can give a number of results at the end. All the commands/orders in the function are within { }. Functions are usually written in a separate file with extension .r and loaded using source(”file.r”). Example:

```
#Normalise function
#Function that normalise a value(s) 'x' in relation to a distribution 'y'
```

```
normalise <- function(x,y) {
  result <- (x-(mean(y,na.rm=T)))/sqrt(var(y,na.rm=T))
  result
}

> ns <- rnorm(n=1000,mean=10,sd=23)
> c(mean(ns),sd(ns))
[1] 10.89364 23.63404
> nn <- normalise(ns[1:100],ns)
> c(mean(nn),sd(nn))
[1] -0.1552580 0.9082879
```

(ii) for(): Programming a loop. A value is going from the first to the last element that is indicated. Example: we want to calculate the value $a_n = \sum_{i=1}^{n-1} \frac{1}{i}$

```
an <- 0
n <- 20
for(i in 1:(n-1)) {
  an <-an + 1/i
}
an
[1] 3.54774
```

(iii) while(): A conditional loop. this loop is only running if the condition inside the parenthesis is true. Example: we will calculate the same value a_n

```
an <- 0
n <- 20
i <- 1
while(i < n) {
  an <- an + 1/i
  i <- i + 1
}
an
[1] 3.54774
```

(iv) if()/else: Conditional function. If the condition inside parenthesis is true, run the next line, otherwise skip this line (or section within { }) and go to *else* { } (if defined). Example:

```
tcoal <- 0
if(runif(1) <= 0.95) {
  tcoal <- 1
}
else {
  tcoal <- -1
}
tcoal
[1] 1
```

2 References

Dalgaard, P. (2002). *Introductory Statistics with R.* Springer.

Everitt, B. and T. Hothorn (2006). *A Handbook of Statistical Analysis Using R.* Chapman and Hall/CRC.