A (very) short introduction to R

1 Introduction

R is a powerful language and environment for statistical computing and graphics. It is a public domain (a so called "GNU") project which is similar to the commercial S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. The main advantages of R are the fact that R is freeware and that there is a lot of help available online. It is quite similar to other programming packages such as MatLab (not freeware), but user-friendlier than programming languages such as C++ or Fortran. You can use R as it is, but for educational purposes we prefer to use R in combination with the RStudio interface (also freeware), which has an organized layout and several extra options. This document contains explanations, examples and exercises, which can also be understood (hopefully) by people without any programming experience. Going through all text and exercises takes about 1 or 2 hours. Examples of frequently used commands and error messages are listed on the last two pages of this document and can be used as a reference while programming.

2 Getting started

2.1 Install R

To install R on your computer, go to the course website and download the version suitable for your computer then install it on your machine

http://hpc.ilri.cgiar.org/beca/training/AdvancedBFX2015/download.html

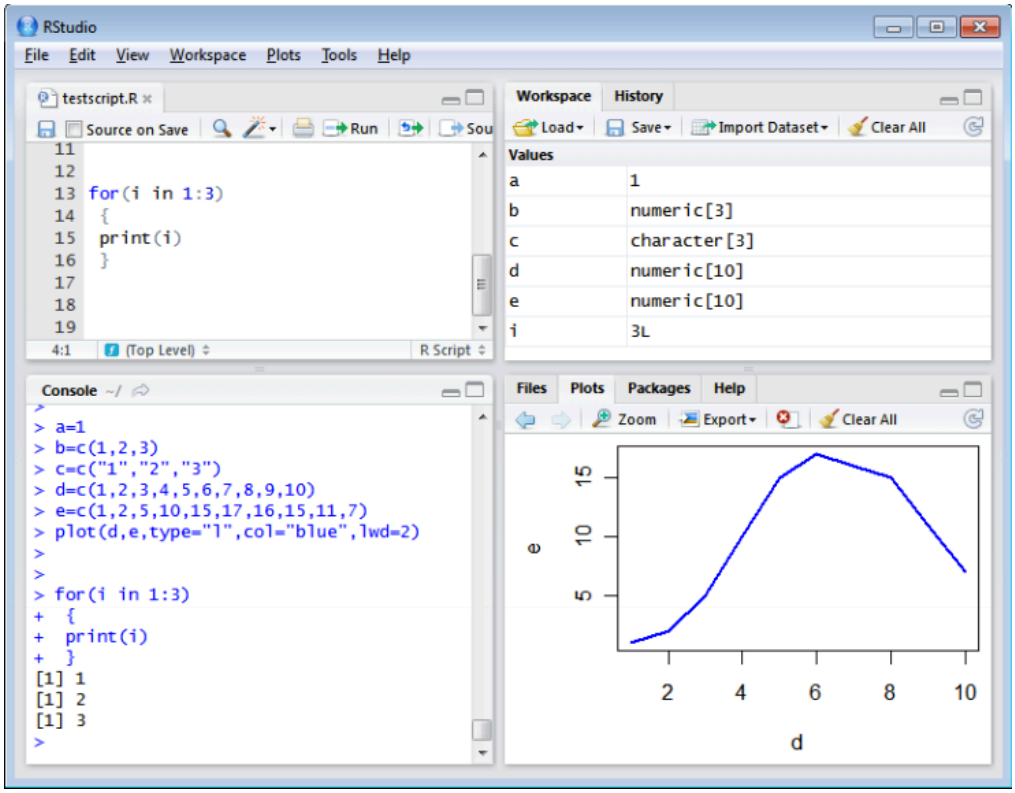
2.2 Install RStudio

To install RStudio, go to the course website and download what is suitable for your Operating system: Install it http://hpc.ilri.cgiar.org/beca/training/AdvancedBFX2015/download.html

2.3 RStudio layout

The RStudio interface consists of several windows (see Figure 1on page2).

- **Bottom left**: console window (also called command window). *Here you can type simple commands after the ">" prompt* and R will then execute your command. This is the most important window, because this is where R actually does stuff.
- **Top left**: editor window (also called script window). Collections of commands (scripts) can be edited and saved. When you don't get this window, you can open it with File → New → R script. Just typing a command in the editor window is not enough; it has to get into the command window before R executes the command. If you want to run a line from the script window (or the whole script), you can click Run or press **CTRL+ENTER** to send it to the command window.
- **Top right**: workspace / history window. In the workspace window you can see which data and values R has in its memory. You can view and edit the values by clicking on them. The history window shows what has been typed before.
- **Bottom right**: files / plots / packages / help window. Here you can open files, view plots (also previous plots), install and load packages or use the help function.



2.4 Working directory

Your working directory is the folder on your computer in which you are currently working. When you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory. Before you start working, please set your working directory to where all your data and script files are or should be stored. Go to **session or tools | Set Working Directory | choose directory**. Then direct it to a folder on you computer

2.5 Libraries

R can do many statistical and data analyses. They are organized in so-called packages or libraries. With the standard installation, most common packages are installed. To get a list of all installed packages, go to the packages window or type **library()** in the console 2 window.

If the box in front of the package name is ticked, the package is loaded (activated) and can be used. There are many more packages available on the R website. If you want to install and use a package (for example, the package called "geometry") you should: Install the package: click install packages in the packages window and type geometry or type **install.packages("geometry")** in the command window. Load the package: check box in front of geometry or type **library("geometry")** in the command window.

3 Some first examples of R commands

3.1 Calculator

R can be used as a calculator you can just type your equation in the command window after the ">":

```
> 10^2 + 36
```

and R will give the answer

```
[1] 136
```

If you use brackets and forget to add the closing bracket, the ">" on the command line changes into a "+". The "+" can also mean that R is still busy with some heavy computation. If you want R to quit what it was doing and give back the ">", press ESC.

3.2 Workspace

You can also give numbers a name. By doing so, they become so-called *variables* which can be used later. For example, you can type in the command window:

```
>a = 4
```

You can see that a appears in the workspace window, which means that R now remembers what a is. You can also ask R what a is (just type and ENTER in the command window):

```
>a
[4]
```

Or do calculations with a

```
>a * 5
[1] 20
```

If you specify a again, it will forget what value it had before. You can also assign a new value to a using the old one.

```
>a = a + 10
>a
[1] 14
```

To remove all variables from R's memory, type

```
>rm(list=ls())
```

or click "clear all" in the workspace window. You can see that RStudio then empties the workspace window. If you only want to remove the variable a, you can type rm(a).

There other ways to define variables lets try the below

```
>a = 4
>a
[1] 4

>b <- 5
>b
[1]5

>assign("c", 6)
>c
[1]6
```

4 Scalars, vectors and matrices

Like in many other programs, R organizes numbers in scalars (a single number – 0-dimensional), vectors (a row of numbers, also called arrays – 1-dimensional) and matrices (like a table – 2- dimensional).

To define a **vector** with the numbers 3, 4 and 5, you need the function c(), which is short for concatenate (paste together).

```
>b=c(3,4,5)
>b
[1] 3 4 5
```

Vectors are the most basic data structures. It is a sequence of data that can either be numbers, characters and also logical.

Elements of vectors must be of the same type and mode. "characters must be enclosed in quotes". NA represents missing data. Do the below to understand vectors more

```
> v = c(4,2,3,8,2,2,5)
> v
[1] 4 2 3 8 2 2 5
> is(v)
[1] "numeric" "vector"
> x = c("a","b","c","d","e")
> x
[1] "a" "b" "c" "d" "e"
> is(x)
[1] "character" "vector"
> y = c(4,2,3,8,2,2,NA,5)
> y
[1] 4 2 3 8 2 2 NA 5
>y*5
[1] 20 10 15 40 10 10 NA 25
```

4.1 Logical vectors

These are vectors with three possible values TRUE, FALSE, NA. They are generated by conditions.

List of logical operators:

- <, <=, >, >=,
- == for exact equality
- != for inequality
- & (and)
- | (or)

```
>x = 1; y = 2
>z = x > y
>z
[1] FALSE
>x<y
>x>y
>x+1 == y
```

To *retrieve or change* the value of an element in the vector use the index of the values.

```
> v = c(4,2,3,8,2,2,5)
> v[4]
[1] 8
> v[4]=10
> v
[1] 4 2 3 10 2 2 5
```

Logical vectors can be used to retrieve data values

```
> z = v<5
> v[z]
[1] 4 2 3 2 2
```

4.2 Factors

Factors are a type of character vector that has its elements defined by groups.

Use factor() on a character vector to create a factor

```
> x = c("groupA", "groupB", "groupA", "groupC",
+ "groupC", "groupB", "groupA", "groupA")
> x = factor(x)
> x
[1] groupA groupB groupA groupC groupC
groupB groupA groupA
Levels: groupA groupB groupC
> levels(x)
[1] groupA groupB groupC
> table(x)
x
groupA        groupB        groupC
        4             2             2
```

Array: multiply subscripted collections of data entries

Matrix: is a 2-diemtional array, matrixes are defined by using the matrix() function or array function.

The **dim()** function can be used to convert a vector to a matrix of data entries

Matrix: is a 2-diemtional array, matrixes are defined by using the matrix() function or array function.

The **dim()** function can be used to convert a vector to a matrix

```
>mat =
matrix(c(4,12,1,5,21,7,10,7,2,19,24,3),
nrow=4, ncol=3)

>mat
         [,1] [,2] [,3]
[1,]  4   21    2
[2,] 12    7   19
[3,]  1   10   24
[4,]  5    7    3

>class(mat)
[1] "matrix"

>mat[2,3]
[1] 19

>x = c(4,12,1,5,21,7,10,7,2,19,24,3)

>dim(x)=c(4,3)
```

Creating matrices using cbind() and rbind(). Arguments to cbind must either be vectors of any length or matrices with same column size, same no of rows

```
>bp = c(132,144,151,120,136)
>ht  = c(183,162,181,168,165)
>wt = c(192,210,240,187,212)
>mat = cbind(bp, ht, wt)
>mat
     bp   ht   wt
[1,] 132  183  192
[2,] 144  162  210
[3,] 151  181  240
[4,] 120  168  187
[5,] 136  165  212
```

4.3 Data frames

The drawbacks of matrices is that all the values have to be the same type. A dataframe is composed of vectors of the same length but different modes.

Specific columns can be accessed using the $ or traditional way of a matrix

- Dataframe$column
- Dataframe[,1]

```
>bp = c(132,144,151,120,136)
>ht = c(183,162,181,168,165)
>wt = c(192,210,240,187,212)
>bg = c("O","O","A","B","AB")
>df = data.frame(bg, bp, ht, wt)
>df
  bg   bp   ht   wt
    1    O  132  183  192
    2    O  144  162  210
    3    A  151  181  240
    4    B  120  168  187
    5   AB  136  165  212

>names(df)
[1] "bg" "bp" "ht" "wt"

>df$bg
[1] O O A B AB

>df$bp[4]
[1] 120

>df[1,4]
[1] 192
>summary(df)

>rownames(df) =
c("p1","p2","p3","p4","p5")
>df
   bg   bp   ht   wt
p1  O  132  183  192
p2  O  144  162  210
p3  A  151  181  240
p4  B  120  168  187
p5 AB  136  165  212
```

Lists: is a collection of objects. It can contain vectors, matrices and data frames of different length

```
>a_list = list(bp, “Swansea Hospital”)
>a_list
[[1]]
[1] 132 144 151 120 136
[[2]]
[1] “Swansea Hospital”

>a_list[[2]]
[1] “Swansea Hospital”

>a_list = list(bloodgroup=bp,
bloodpress=bp, height=ht, weight=wt,
hospital=“Swansea Hospital”,
doctors=c(“Dr. Lewis”, “Dr. Hill”)
>a_list
$bloodgroup
[1] “O” “O” “A” “B” “AB”
$bloodpress
[1] 132 144 151 120 136
$height
[1] 183 162 181 168 165
$weight
[1] 192 210 240 187 212
$hospital
[1] “Swansea Hospital”
$doctors
[1] “Dr Lewis” “Dr. Hill”
```

4.4 Functions

These contain pre-written code that performs some task; e.g. Sum is a function that will give the sum, mean, rnorm generates random numbers

```
>sum (bp)
[1] 683

>mean (bp)
[1] 136.6

>rnorm(8)
```

5 Plots

R can make graphs the following is a simple example

```
>x = rnorm(100)
>plot(x)
```

In the first line 100 random numbers are assigned to variable x, which becomes a vector for this plot.
In the second line, all these values are plotted in a scatter plot in the plot window.
Types of plots;
“p” – points
“l” – lines
“b” – both – lines and points
“c” – lines but not where points are
“o” – non overlapping points and lines
“h” – histogram-like vertical lines
“s” – draw lines as steps
“n” – draw only axes.

```
>plot(rnorm(100), type=“l”, col=“gold”)
```

Try and change the type of plot severally and see what you get.
Some more useful graphical arguments are also
lty – line type (dashed, solid, dotted, etc)
lwd – line width
col – color of the plotted points
pch – style of points (circle, cross, start)
cex – scaling of point size and text
title() – add a title and subtitle
legend() – add a legend
xlab – label for x-axis
ylab – label for the y-axis

```
>plot(c(124,118,130,127,103,141,114),
c(75,80,95,77,68,105,84),xlab="systolic",
ylab="diastolic", xlim=c(90,150),
ylim=c(60, 110), lwd=3, col=”blue”, cex=4,
pch=20)
```

Histogram plot: another very simple example is the classical histogram plot

```
>hist(rnorm(100))
```

Bar charts: draws a bar with height proportional to the count in a table. Let's make a barplot of both frequencies and proportions. First, we use the scan function to read in the data then we plot. We will use the table command to create summarized data, which is then bar

```
>y=scan()
3 4 1 1 3 4 3 3 1 3 2 1 2 1 2 3 2 3 1 1 1 1 4 3

>barplot(table(y))

>barplot(table(y)/length(y))
```

plotted

Pie Charts: the same data can be studied using pie function

```
>z=(table(y))

>pie(z)

>names(z) = c("first", "second","third","fourth")

>pie(z)

>pie(z, col=c("purple", "green", "red", yellow))
```

6 More on R packages

Bioconductor is an open source R software for bioinformatics. A lot of the tools needed for bioinformatics are available at within bioconductor

6.1 How to install bioconductor packages

```
>source("http://bioconductor.org/bioclite.R")
>biocLite("name of package")
```

7 Some useful references

7.1 Functions

This is a subset of the functions explained in the R reference card.

Data creation

- **read.table**: read a table from file. Arguments: header=TRUE: read first line as titles of the columns; sep=",": numbers are separated by commas; skip=n: don't read the first n lines.
- **write.table**: write a table to file
- **c**: paste numbers together to create a vector
- **array**: create a vector, Arguments: dim: length
- **matrix**: create a matrix, Arguments: ncol and/or nrow: number of rows/columns
- **data.frame**: create a data frame
- **list**: create a list
- **rbind and cbind**: combine vectors into a matrix by row or column

Extracting data

- **x[n]**: the n th element of a vector
- **x[m:n]**: the mth to nth element
- **x[c(k,m,n)]**: specific elements
- **x[x>m & x<n]**: elements between m and n
- **x$n**: element of list or data frame named n
- **[i,j]**: element at ith row and jth column
- **[i,]**: row i in a matrix

Information on variables

- **length**: length of a vector
- **ncol or nrow**: number of columns or rows in a matrix
- **class**: class of a variable
- **names**: names of objects in a list
- **print**: show variable or character string on the screen
- **is.na**: test if variable is NA
- **as.numeric or as.character**: change class to number or character string

Statistics

- **sum:** sum of a vector (or matrix)
- **mean**: mean of a vector
- **sd:** standard deviation of a vector
- **max or min**: largest or smallest element
- **rowSums** (or rowMeans, colSums and colMeans): sums (or means) of all numbers in each row (or column) of a matrix. The result is a vector.
- **quantile(x,c(0.1,0.5))**: sample the 0.1 and 0.5th quantiles of vector x

Data processing

- **seq**: create a vector with equal steps between the numbers
- **rnorm:** create a vector with random numbers with normal distribution
- **sort:** sort elements in increasing order
- **t:** transpose a matrix
- **na.approx**: interpolate (in zoo package). Argument: vector with NAs. Result: vector without NAs.
- **cumsum**: cumulative sum. Result is a vector.
- **rollmean**: moving average (in the zoo package)
- paste: paste character strings together
- **substr**: extract part of a character string

Plotting

- **plot(x)**: plot x (y-axis) versus index number (x-axis) in a new window
- **plot(x,y)**: plot y (y-axis) versus x (x-axis) in a new window
- **image(x,y,z)**: plot z (color scale) versus x (x-axis) and y (y-axis) in a new window
- **lines or points**: add lines or points to a previous plot
- **hist**: plot histogram of the numbers in a vector
- **barplot**: bar plot of vector or data frame
- **contour(x,y,z)**: contour plot

- **abline**: draw line (segment). Arguments: a,b for intercept a and slope b; or h=y for horizontal line at y; or v=x for vertical line at x.
- **curve**: add function to plot. Needs to have an x in the expression. Example: curve(x^2)
- **legend**: add legend with given symbols (lty or pch and col) and text (legend) at location (x="topright")
- **axis**: add axis. Arguments: side – 1=bottom, 2=left, 3=top, 4=right
- **mtext**: add text on axis. Arguments: text (character string) and side
- **grid**: add grid
- **par**: plotting parameters to be specified before the plots. Arguments: e.g. mfrow=c(1,3)): number of figures per page (1 row, 3 columns); new=TRUE: draw plot over previous plot.

Keyboard shortcuts

There are several useful keyboard shortcuts for RStudio (see Help → Keyboard Shortcuts):

- **CRL+ENTER**: send commands from script window to command window
- ↑ **or** ↓ in command window: previous or next command
- **CTRL+1, CTRL+2**, etc.: change between the windows

Not R-specific, but very useful keyboard shortcuts:

- **CTRL+C, CTRL+X and CTRL+V**: copy, cut and paste

- **ALT+TAB**: change to another program window

- **↑, ↓, ← or →**: move cursor

- **HOME or END**: move cursor to begin or end of line