Introduction to R

Andrew McAdam and Karl Cottenie

September 13, 2011

Contents

Welcome to R! This is the script for my lecture on an introduction to R. In this script you will find the code that I plan to use in lecture as well as the output from R as it appears on my screen. This script will provide in a lot of cases both the input and output of the R session. I recommend that as you work your way through the script, you type the code into R, instead of using copy-paste. This will allow you to get used to typing R code, and get used to trouble-shooting any error messages you will get.

These scripts are based on the course notes of Andrew McAdam, and adapted/changed by Karl Cottenie. So when you read "I", you have to initially put Andrew's voice in your head, and not mine (Karl's), most of the time.

I will provide both a pdf and an html version of the code. I am really interested in what format you like best for reading and working with the code. Just let me know any comments you might have related to the format, pace of delivery, etc.

1 Interactive interface

1.1 Calculator

The first thing to note is that the "R console" represents your interface with R. This is a Question and Answer type interface where you submit a command and often recieve some sort of output from R. The ">" symbol indicates that R is ready to receive an input from you. In order to add comments to my R session you can see that I have preceded this text with the "#" character. This tells R to ignore what follows as it is only a comment.

If I forget to enter the # character I will get an error message because R doesn't understand this sentence.

I have entered some additional lines to make the document a little easier to read.

One of the first things to consider is the directory in which you are currently working. You can ask R for your current Working Directory using:

```
getwd()
```

```
 [1] "/Users/karlcottenie/Desktop/1. Teaching/courses/2012 FAR/2011-IBIO6000/Topics/To
```

You can change the working directory using:

```
# setwd()
```

Note that R is case-sensitive so that Setwd and setwd are not equivalent

```
Getwd()
```

```
 Error: could not find function "Getwd"
```

Apparently there is no function called "Getwd"

Both comments and commands can wrap around more than one line, but a carriage return ends both. If you command wraps around more than one line you will receive the + prompt instead of the prompt indicating that R is waiting for more information from you.

One of the simplest thing that R can be used for is simple calculations

```
5+18
```

```
 [1] 23
```

Note that the output of the command is preceded by numbers in square brackets. This is just an index for keeping track where the answer was put. It actually means that it is the first value in a vector.

```
10*230
```

```
(10+10)/5+6
```

```
#versus
```

```
(10+10)/(5+6)
```

```
sqrt(81)

log(10)

#Note that this is different from

log10(10)

exp(1)
```

```
 [1] 2300
 [1] 10
 [1] 1.818182
 [1] 9
 [1] 2.302585
 [1] 1
 [1] 2.718282
```

Note that the "log" and "exp" functions refer to base e and not base 10.

1.2 Storing information

In addition to receiving the output directly, the result of a calculation can be stored as a new variable.

```
x<-5+18

#or

x=5+18
```

We can recall this variable whenever we need to...

```
 x

 #Note that we can call the variable pretty much whatever we want
 answer<-5+18

 answer
```

```
 y=10*4

y

 z=x*y

z

 [1] 23
 [1] 23
 [1] 40
 [1] 920
```

Note that "=" or "<-" assigns a particular value to a variable. We can also ask a question.

```
z==x

 [1] FALSE
```

In this case we have asked R whether z is equal to x. It has told us that this expression is FALSE. NOTE THAT THIS IS VERY DIFFERENT FROM:

```
z=x

#or z<-x

z

x

z==x

 [1] 23
 [1] 23
 [1] TRUE
```

1.2.1 Vectors

Instead of creating a variable we can also create vectors.

```
variable <- c(12, 33, 45, 101, 65, 30, 55, 99, 70, 84)

#The c is short for concatenate

variable

#Note again that the [1] just helps us keep track of the fact that the value 12 is the

one.to.ten <- 1:10

one.to.100 <- 1:100

one.to.100

#also

one.to.ten <- seq(1,10)

one.to.ten

#or

one.to.ten <- seq(1,10, 2)

one.to.ten
```

```
  [1]  12  33  45 101  65  30  55  99  70  84
   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  [91]  91  92  93  94  95  96  97  98  99 100
  [1]  1  2  3  4  5  6  7  8  9 10
 [1] 1 3 5 7 9
```

Note that the vector now wraps around more than one line and so the [] values at the left help us to keep track of where we are in the vector.

Just like variables, you can also do math with vectors

```
one.to.100*5
```

```
  [1]   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
 [19]  95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180
 [37] 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270
 [55] 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360
 [73] 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450
 [91] 455 460 465 470 475 480 485 490 495 500
```

Because R is a statistical program you can also easily generate random numbers from a known distribution

```
random<-rnorm(25, mean=0, sd=1)

random

plot(random)
```

```
  [1] -0.81917338  0.97522976 -0.61860196 -0.44771966 -0.36653009 -1.26314677
  [7]  0.25191421  0.10662756  1.19632510  0.88499114 -0.72279693 -0.90327264
 [13] -0.74869131 -0.03359329  0.29886874 -0.44107791 -0.84547728 -0.37608161
 [19]  0.43212531  3.20605895 -1.22445281  0.70142745  0.61701119  0.54181404
 [25] -0.68355117
```

The plot function above generates a new graphical interface. R has a great deal of flexibility and power in how it generates graphics. You can do fairly simple things as we did above, but you can also get quite detailed in yoru graphics. I will spend more time talking about graphics later.

We can also check that this vector has 25 elements, a mean of zero and a sd of one.

```
 length (random)

sum(random)/length(random)

 mean.random<-sum(random)/length(random)

sqrt(sum((random-mean.random)^2)/(length(random)-1))

 #or simply

 mean (random)

 sd(random)
```

```
 [1] 25
 [1] -0.01127093
 [1] 0.975954
 [1] -0.01127093
 [1] 0.975954
```

1.2.2 Functions

We have already encountered a number of R functions already (e.g., log(), mean(), sqrt(), etc.). In addition to creating objects, variables and vectors, we can also write functions in R that will help to simplify life later. Remember that we already looked at:

```
log(exp(10))
```

```
 [1] 10
```

As we saw there is also a log10(x) function, but no exp10(x) function so let's write one!

```
exp10(10)
```

```
exp10<-function(x){
#This function accepts a value, x, as input and returns the value of 10 raised to the p
10^x
}
```

```
 [1] 1e+10
```

We can now recall our function

```
exp10
```

```
 function(x){
 #This function accepts a value, x, as input and returns the value of 10 raised to the
 10^x
 }
```

As a test of our function...

```
exp10(3)
```

```
 [1] 1000
```

This function only contains one argument but we can easily make changes so that it contains several

```
 exp10<-function (x, y){
#This function is bogus but calculates 10 raised to the power of x but adds some other
 10^x+y
 }

 exp10(3, 4)

 #Note that if I fail to include a necessary parameter then I will get an error

 exp10(3)

 #This isn't a very useful function so I am going to change it back

 exp10 <- function(x){
#This function accepts a value, x, as input and returns the value of 10 raised to the p
 x <- 10^x
x
 }

 [1] 1004
  Error in 10^x + y : 'y' is missing
```

The other important thing to know about functions is that they are like Las Vegas. What happens in the function stays in the function. That is, we have defined a variable x within the function. This variable is used for the purpose of the function but does not affect any variable that we might have in our workspace called x

```
 x <- 1
 temp <- exp10(3)
 x
temp

 [1] 1
 [1] 1000
```

2 Scripting interface

2.1 Definition

R stores variables, datafiles, functions, vectors, etc in what is called the Workspace. This contains all of the items that you can access directly within your R session. You can list all of the objects in your workspace using:

```
ls()
```

```
  [1] "answer"      "ashina"      "exp10"       "mean.random" "one.to.100"
  [6] "one.to.ten"  "random"      "temp"        "variable"    "x"
 [11] "y"           "z"
```

Let's say I didn't want to keep the variable vector around. I can delete it from the workspace using

```
rm(variable)

#It is now erased!!

variable

ls()

#I can delete more than one object using

rm(answer, one.to.100, random)

ls()
```

```
  Error: object 'variable' not found
  [1] "answer"      "ashina"      "exp10"       "mean.random" "one.to.100"
  [6] "one.to.ten"  "random"      "temp"        "x"           "y"
 [11] "z"
 [1] "ashina"      "exp10"       "mean.random" "one.to.ten"  "temp"
 [6] "x"           "y"           "z"
```

2.2 Saving the workspace

It is a good idea to save your workspace so that variables, etc can be used in a later session

```
save.image()

#This will save your workspace as your default workspace.  You can also save it to a pa

save.image("first workspace")
```

If you are working in R and you want to load a particular workspace then you can load it using

```
#load(".RData")
```

Before quitting R you will be asked whether you want to save your workspace. This refers to the default workspace.

2.3 Saving the console/code

There are two ways of interacting with R:

- interactive mode. This is how we have used it so far. The advantage is that you can check how things work on the fly. The disadvantage is that you do not have a record of what you have done, explicitly. You can save this by saving the console, by using the menus. In that case, you save the console as a .txt file, that you can open for instance in Word. Convert the font into Courrier or Courrier New to make sure that the alignment is preserved.

- script editor mode. In this case, you write all your code in a text file (save it as .R for instance), and you copy-paste the code into the console. At the end of the day, you just have to save your workspace and your script, and you have a complete document of all your steps leading to an analysis.

For this course, I will "force" you to use the script editor mode, since I want you to create a reproducable document, and that is the easiest way to obtain this. However, you can also try short pieces in the console, before you put the final version in the script editor. With that in mind, the most often way to organize your interface with R, is to have your script editor in the top part of your screen, and the console editor in the bottom part. This is how the 2 interfaces I recommend (and Emacs which is what I use) work.

3 Getting help

There are a number of places where you can get help with R directly from the console.

```
#?lm
```

This brings up a description of the function "lm"

```
help.search("procrustes")
```

```
Help files with alias or concept or title matching 'procrustes' using
fuzzy matching:


ade4::procuste          Simple Procruste Rotation between two sets of
                        points
vegan::procrustes       Procrustes Rotation of Two Configurations and
                        PROTEST


Type '?PKG::FOO' to inspect entries 'PKG::FOO', or 'TYPE?PKG::FOO' for
entries like 'PKG::FOO-TYPE'.
```

This brings up all references to the lm function in packages and commands in R. We will talk about packages later.

```
RSiteSearch("procrustes")
```

```
 A search query has been submitted to http://search.r-project.org
 The results page should open in your browser shortly
```

A search query has been submitted to http://search.r-project.org The results page should open in your browser shortly.

This is quite a comprehensive search that covers R functions, contributed packages and R-help postings. It is very useful but uses the web.

4 Packages

4.1 Reference to R

You will often need to make reference to this statistical package (i.e. in your final project, thesis and publications). There is no manual to directly reference. Instead R provides a reference for you to use. In case you ever forget how to find it look at the reminded that is given to you above. . . "citation()"

```
citation()


To cite R in publications use:

  R Development Core Team (2011). R: A language and environment for
  statistical computing. R Foundation for Statistical Computing,
  Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Development Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2011},
    note = {{ISBN} 3-900051-07-0},
    url = {http://www.R-project.org/},
  }

We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also 'citation("pkgname")' for
citing R packages.
```

4.2 Packages

R is just one big collection of packages. Some of these are so essential to the functioning of R that they are automatically loaded when R boots up. Others have to be loaded when you need them. Still others don't come immediately downloaded with R. You will, therefore, need to download these packages before loading them

You can see a list of all of the available packages on the CRAN site

****** Packages need to be installed only once, but THEY MUST BE LOADED EACH TIME YOU OPEN R. ********

```
ISwR
ashina

Error: object 'ISwR' not found
   vas.active vas.plac grp
```

```
1         -167     -102   1
2         -127      -39   1
3          -58       32   1
4         -103       28   1
5          -35       16   1
6         -164      -42   1
7           -3      -27   1
8           25      -30   1
9          -61      -47   1
10         -45        8   1
11         -38       12   2
12          29       11   2
13           2       -9   2
14         -18       -1   2
15         -74        3   2
16         -72      -36   2
```

ISwR is a package of datasets that was put together by Dalgaard to go along with your book. We can load this package in the menus or with the library() command.

```
#install.packages("ISwR") #if necessary, remove the # at the start of
this line
library(ISwR)
```

```
 Error: unexpected symbol in "this line"
```

Note that ISwR hasn't added anything to my workspace, but functions and datasets within the package can now be accessed from the console.

We can load a datafile called "ashina" from ISwR

```
data(ashina)
```

```
ashina
```

```
   vas.active vas.plac grp
1         -167     -102   1
2         -127      -39   1
3          -58       32   1
4         -103       28   1
5          -35       16   1
```

```
6        -164      -42   1
7          -3      -27   1
8          25      -30   1
9         -61      -47   1
10        -45        8   1
11        -38       12   2
12         29       11   2
13          2       -9   2
14        -18       -1   2
15        -74        3   2
16        -72      -36   2
```

The data file ashina is now part of the workspace and can be modified or used for analyses just like any other object.

```
ls()
```

```
 [1] "ashina"      "exp10"       "mean.random" "one.to.ten"  "temp"
 [6] "x"           "y"           "z"
```

4.3 Groups of packages

The multitude of R packages quickly resulted in the problem of package and function detection. R solves this problem partly by "Task Views". Each Task View has a maintainer who is an expert in a particular field of science, e.g., multivariate statistics, or spatial analyses. The maintainer than provides an overview of the packages that could be useful in that field. The advantage of using Task Views is that it not only provides this overview of the different packages, but also an easy way to install all these packages in one step. You first install the "ctv" package (install.packages("ctv"), and load the package (library("ctv"). Second, you install the packages within a specific Task View by install.views("Environmetrics"), or update the packages by update.views("Environmetrics").