R language

Universidade do Porto

Faculdade de Engenharia

FEUP

A LANGUAGE FOR DATA ANALYSIS

HTTP://WWW.R-PROJECT.ORG/

Table of contents

- Basic concepts
- R as a calculator
- Objects
  - Variables
  - Vectors
  - Matrices
  - Arrays
  - Lists
  - Dataframes
  - Additional functions on objects
- Importing/exporting data
- Interaction with the user
- Control structures
  - Conditional statements
  - Iterative statements
  - Functions with implicit iterative processes
- Functions
- Help
- Bibliography

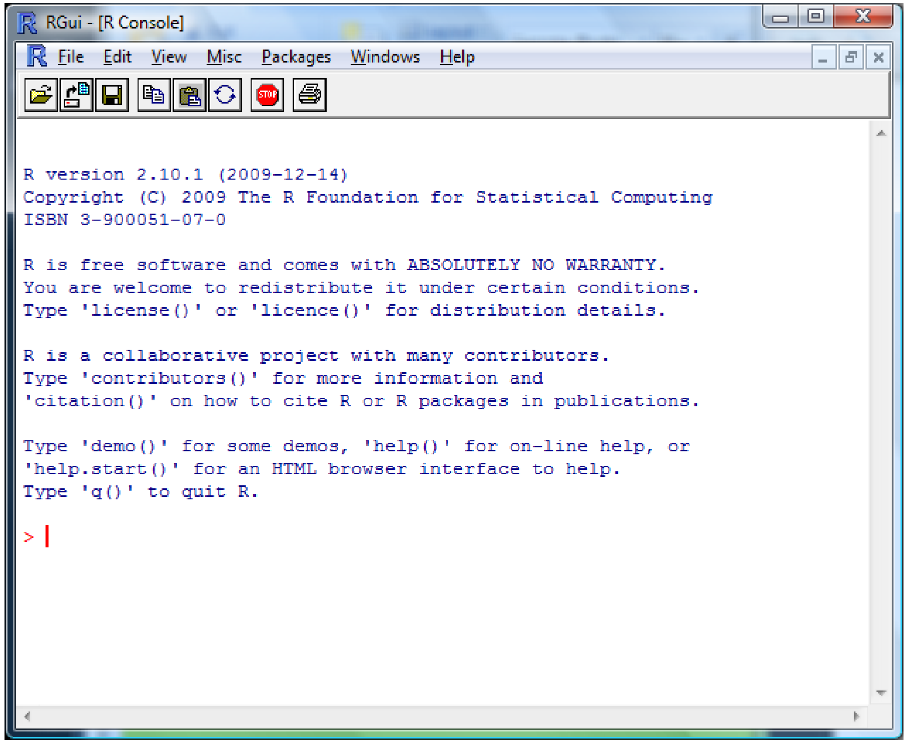
Basic concepts

ÅR is a free *software* for computational statistics, data analysis, data mining, and much more é

ÅDownload: http://cran.dcc.fc.up.pt/

ÅYou can do the base installation and install further packages later

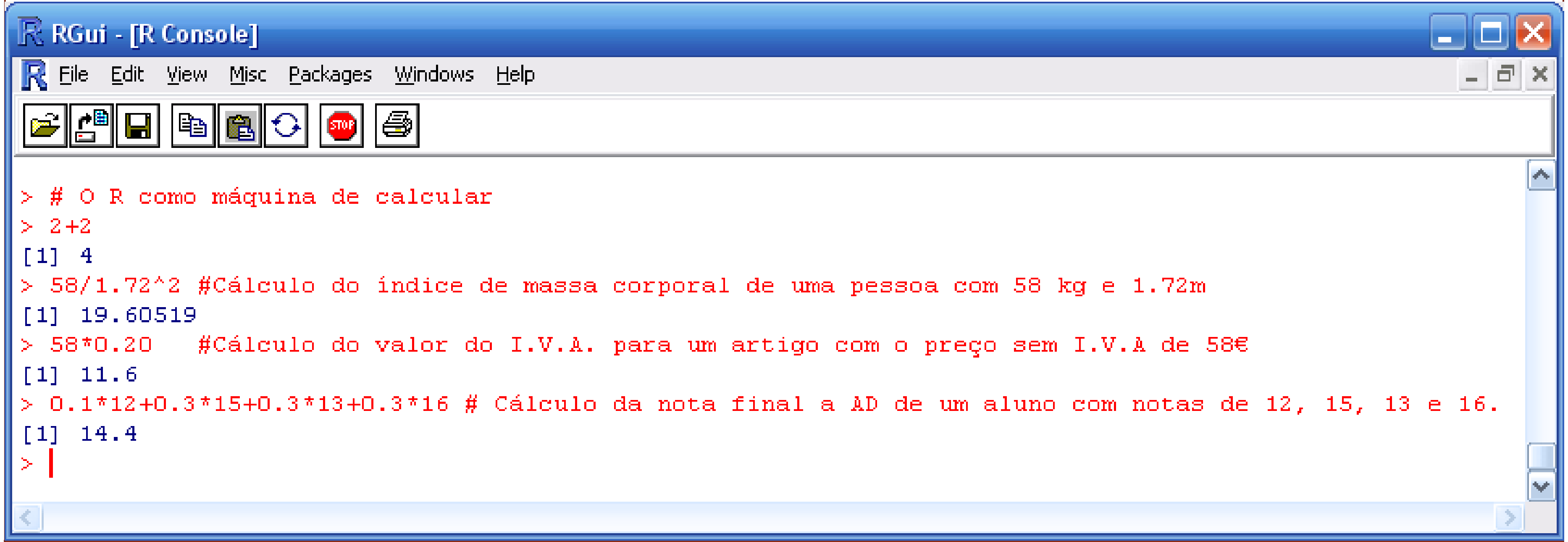
O ambiente de trabalho



```
R version 2.10.1 (2009-12-14)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

R as a calculator

In red: the instructions
In blue: the results
#: *line comments*
/ comments of several lines */*



```
> # O R como máquina de calcular
> 2+2
[1] 4
> 58/1.72^2 #Cálculo do índice de massa corporal de uma pessoa com 58 kg e 1.72m
[1] 19.60519
> 58*0.20   #Cálculo do valor do I.V.A. para um artigo com o preço sem I.V.A de 58€
[1] 11.6
> 0.1*12+0.3*15+0.3*13+0.3*16 # Cálculo da nota final a AD de um aluno com notas de 12, 15, 13 e 16.
[1] 14.4
> 
```

Objects

- Variables
- Vectors
    - Sequences
    - Factors
- Matrices
- Arrays
- Lists
- Dataframes
- Additional funcions

Objects: variables

It is possible to save values or results of operations on variables.

Try the following:

```
x<-2 # <- assigns a value to a variable
x      # The names of variables are case sensitive: x e X are different objects.
x+x
x<-x+x
x
text<-'hello'   # A variable can store non-numeric values
```

Objects: vectors

Vectors are ordered sets of values:

```
weight        <- c(60, 72, 57, 90, 95, 72)
height        <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
bmi           <- weight/height^2                    # bmi: body mass index
names(bmi)    <- c('Ana','Rui','Isabel','Paulo','Eva','Diogo')
bmi
length(bmi)   # vector size
typeof(weight) # datatypes: ñlogical", "integerñ "double", "character", "NULL", é
weight        <- as.integer(weight) #changing datatype to integer
```

Sequences are vectors of non-negative integers;

```
x <- 0:10
x
seq(0,10,1)
seq(1,10,2)
```

Factors are vectors having enumerable values, i.e., it is a finite set:

```
classif <- factor(c('insufǫ́'sufǫ́'insuf','bom','suf'))
levels(classif)
levels(classif) <- c(levels(classif),'muito bomǫ̉
classif
```

Objects: vectors

You can use indexes to access the values of the vectors. Try The Following:

```
bmi[3]
bmi[-3]
bmi[1:3]
bmi[c(1,3)]
bmi["Rui"]
bmi[bmi>=22.5]
bmi[bmi>=20 & bmi <=25]          # & is the logical operator AND
bmi[bmi<20 | bmi>25]             # | is the logical operator OR
```

The **function *order*** read the indexes of the vectors by a given order of their values:

```
idx<-order(bmi)                  # increasing order by default
idx
bmi[idx]
order(bmi, decreasing=TRUE)      # decreasing order
```

The **function *sort*** orders the vector according to a given order (increasing or decreasing):

```
sort(bmi)                        # by default the order is increasing
sort(bmi, decreasing=TRUE)       # decreasing order
```

Exercises with R: vectors

1. A company has 2 branches: one in Porto another in Lisboa. Monthly revenues (in EUR ' 000) in each of the agencies were:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Porto	35	38	40	38	42	37	33	25	36	39	40	45
Lisboa	62	70	74	76	75	65	58	50	70	73	75	78

Answer using the R language:

a) Keep this information in a proper manner and build an object containing the global revenue for each month of the year.

b) Which months whose global revenue was less than 100 thousand euros?

c) Which months whose global revenue was less than 100 thousand euros or more than 120 thousand euros?

d) Which months whose global revenue was greater than 100 thousand euros and less than 120 thousand euros?

e) Order the months by descending order of the overall revenue.

2. Build a sequence of odd numbers between 0 and 20.

3. Construct a vector with the grades obtained in the first semester and give the names of disciplines to those grades.

Objects: matrices

Matrices can be viewed as vectors of two dimensions::

```
grades <- matrix(c(15,17,15,16,15,18,15,16,12,17,14,12),3,4)
grades                        # pay attention to the order by which the matrix is fullfilled
rownames(grades) <- c('Mário','Lúcio','Amaro')
colnames(grades) <- c('SI','EC','AD','IA')
grades
grades <- rbind(grades, c(15,13,13,14)) # it adds a row
grades
grades <- grades[-nrow(grades),]          # ~~it adds a column~~
grades
```

Number of rows

Number of columns

Indexing of arrays is similar to vector indexing:

```
grades[1,2]
grades[1,]
grades[,2]
grades['Amaro','AD']
```

Objects: arrays

The **arrays** can be seen as matrices with more than two dimensions. Example of an array with 3 dimensions::

```
grades <- array(c(15,17,15,16,15,18,15,16,12,17,14,12,
                  13,14,15,16,14,12,15,16,10,15,14,12), c(3,4,2))
grades          # pay attention to the order by which the matrix is fullfilled
dimnames(grades)      <- list(c('Mário','Lúcio','Amaro'), c('SI','EC','AD','IA'),
                              c('ExtraTime','RegularTime'))
grades
```

Indexing arrays is similar to indexing matrices:

```
grades[2,3,2]
grades[,2,]
grades[1,2,]
grades['Amaro', 'AD', 'ExtraTime']
```

Objects: lists

A **list** is a set of ordered objects. The listôs objects can be of different types: variables, vectors, matrices, etc.

```
student1 <-  list(name='Mário', course='CEI', grades=c(15,16,15,17))
student2 <-  list(name='Lúcio', course ='CEI', grades=c(17,15,16,14))
student3 <-  list(name='Amaro', course ='CEI', grades=c(15,18,12,12))
students <-  list(student1, student2, student3)
students
```

Indexing lists:

```
students[[1]]
students[[1]]$name
students[[2]]$grades[3]
students[[1]]$scholar.year <- 3 # adds óscholar.year' to student1
students
```

Exercises with R: matrices

1. Add the discipline of Software Engineering (ES) to the matrix *grades* knowing that the grades Mário, Lúcio and Amaro were 16, 17 and 15, respectively. Use the *cbind* (is identical to the *rbind* function, but for columns).
2. Present all grades of Lúcio.
3. Build a matrix with the distances between Lisboa, Porto and Guarda, knowing that Porto is 317 km from Lisboa and 203 km from Guarda and Lisboa is 317 km away from Guarda.

Objects: dataframes

A **dataframe** is a data structure in table format. Unlike matrices, the data types are defined per column:

```
students <-  data.frame(name=c('Mário', 'Lúcio', 'Amaro'),
                 course=c('CEI', 'CEI', 'CEI'), SI=c(15,17,17),
                 EG=c(16,15,18), AD=c(15,16,12), IA=c(17,14,12))
students
```

Indexing dataframes:

```
students[1,3]
students[students$name=='Mário', 'SI']
students[students$name=='Mário',]$SI
students[students$AD<15,]
students[students$SI<15 | students$EG<15 | students$AD<15 | students$IA<15,]
attach(students)
students[SI<15 | EG<15 | AD<15 | IA<15,]
```

Objects: dataframes

```
students <-  rbind(students,c('António', 'CEI',14,14,13,15)) # adds a row
Warning message:
In `[<-.factor`(`*tmp*`, ri, value = "António") :
  invalid factor level, NAs generated
students$name <- as.character(students$name)
levels(students$name)<-c(levels(students$name), 'António')
students[4,]$name <- 'António'
students$name<-as.character(students$name)
students <-  rbind(students,c('Celso', 'CEI',15,13,13,14)) # adds another row
length(which(students$AD>14))
students <- cbind(students, c(16,17,15,15,14)) # adds another column
colnames(students)[ncol(students)] <- 'ES'
nrow(students) # number of rows
```

Difficult? An easier way ...

```
students <- edit(students)
```

Exercises with R: dataframes

1. Which students had a grade greater than 14 at AD?
2. How many grades greater than 14 were obtained in AD?
3. Which are the grades obtained by Lúcio?
4. Who had grades between 16 and 18, inclusive?
5. What are the names of the students and their grade to SI considering only the students who had a grade greater than 14 to AD?

Objects: additional functions

ls(): lists all existing objects.

rm(obj1, obj2, …): remove the object(s) specified.

Importing/exporting data

On importing

In the R menu: File -> Change dir … → In that folder the <*file*>*.csv should exist*. You can create ir with Excel …

```
df <- read.csv('<file>.csv', sep = ",")
```

This symbol must be the one used in file <*ficheiro*>*.csv* to separate columns. Open the file and verify which is the symbol used…

On exporting

There are numerous methods for exporting R objects into other formats. Example for the .csv format:

```
write.table(mydata, “<file>.txt”, sep=“,”)
```

Interaction with the user

Writing in the ecran

Å*print:* to write any object in the ecran.

Å*cat:* it uses an arbitrary number of arguments. Convert the arguments to strinngs, append them, and write the result in the ecran.

```
he <- óJoãoô
money <- 150
cat(he, ñwins ñ, money, ñeuros, the poor!\nò)
```

Interaction with the user

Reading data

Å*scan:*

```
> x<-scan(n=5)
1: 45 66 34.2 456.7 7
> x<-scan()
1: 45 66 34.2
4: 456.7 7
6: 12.2
7:
> x<-scan(what=character())
1: óAdrianaô óBrunoô
3: óCíntiaô óJoãoô óLiaô óWálterô
7:
```

Control structures: conditional statements

The *if* statement

Try the following instructions:

```
> if ((x<-scan(n=1))==1) cat(‘Hello\n') else cat(‘Bye\n')
> if (x > 0) {cat('x is positive.\n')
y <- 10 * x}     # Example of using if without else
> if (age < 18) {
group <- 1
} else if (age < 35) {
group <- 2
} else if (age < 65) {
group <- 3
} else {
group <- 4 } # Example of nested ifs
```

Control structures: conditional statements

The *ifelse* conditional statement

Its use with vectors:

```
x <- c(10,15,8,13,5,19,16,14,10)
ifelse(x<10,'reprovou','passou')
```

Control structures: iterative statements

The cycle *repeat*

```
repeat
<statements block>

pos<- c()
repeat {
        cat(óintroduce a positive number ? (zero finishes) ')
        nr <- scan(n=1)
        if (nr < 0) next       # next ignores the value and continues
        if (nr == 0) break  # break exists the cycle.
        pos <- c(pos,nr)
}
```

Control structures: iterative statements

The cycle *while*

Cycles of the type: while *condition is TRUE do*.

```
while (<Boolean condition>)
<statements block>

i <- 1
n <- 5
res <- 1
while (i<=n) {
	res <- res*i
	i <- i+1
}	# which function is this one?
```

Control structures: iterative statements

The cycle *for*

Cycles of the type: For *values rang* do.

```
for(<var> in <set>)
<statements block>
n <- 5
res <- 1
for (i in 1:n) {
        res <- res*i
}       # which function is this one?
```

Control structures

Functions with embbeded iterative processes

<u>The function *apply*</u>

It applies a given function to all ros or columns of a matrix, array or dataframe.

```
apply(<data>,<1 or 2>, <function>)
# If 1, apply the function to rows; If 2, applies the function to columns.
data(iris)
apply(iris[,1:4],2,mean)
```

<u>The function *tapply*</u>

It applies a given function according to a given aggregation criterion.

```
tapply(<data>,<aggregation criterion>, <function>).
data(warpbreaks)
tapply(warpbreaks[,1], warpbreaks[,2],mean)
tapply(warpbreaks[,1], warpbreaks[,2:3],mean)
```

Control structures

Functions with embbeded iterative processes

The function *sapply*

It applies a function defines by the user to each value of data.

```
sapply(<data>, <function>)
sapply(iris[,1], function(y) (y - mean(iris[,1]))/sd(iris[,1]))
# The function will be calculated for each one of the elements of iris[,1], i.e., the y
```

Functions

How to create functions in R:

    <name of the function> <- function(<parameters>) { <statements block> }

```
coef.var <- function(x,i)
{
    res <- (x[i]-mean(x))/sd(x)
    res
}
coef.var(iris[,1],37)
sapply(iris[,1], function(y)(coef.var(as.vector(iris[,1]),y)))
```

Functions

```
factorial <- function(n)
{
        res <- 1
        for (i in 1:n) res <- res*i
        res
}
factorial(5)

factorial <- function(n, res=1) # recursive version of factorial
# by default res=1.
{
        if (n >0) res <- factorial(n-1, res*n)
        res
}
factorial(5)
```

Exercises with R: functions

1. Create a function that, given the dataframe iris and one of its varieties (setosa, versicolor or virginica), returns a vector with the percentage of that variety.
2. Create a function that receives as argument, the name of a csv file, open it to a dataframe and, write in the ecran the number of numerical attributes it has (use the function is.numeric).
3. Given the dataset traveltime78 create a function that having the dataset set ordered by data defines a 5 day window and calculates the average of the Duration for that window. Sliding the window one day it calculates the average again. This should be done until the end of the dataset and the result of the averages for each window should be stored and returned in a vector.

Help

help.start() # initial web page with the documentation of R

help(*lm*) # help about a function, in the example: *lm*. Similar to ?*lm*

```
lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x =
    FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)
```

# The help about functions uses <var> = <value> with the meaning that, by default,
# <var> uses as value <val>

help(package=*stats*) #help about, in this example, the package *stats*

help.search('regression') # search functions related to a given subject, in this example, *regression*. Equivalent to *??regression*. It does a wider search than *help()*.

apropos('*lm*'*)* # function that have, in this example, *lm* in the name

example(*lm*) # executes the exemples that are in the help page of that function. In this example, *lm*

Bibliography

Introductory texts to R:

- Luís Torgo, "Introdução à programação em R" (in portuguese), http://cran.r-project.org/doc/contrib/Torgo-ProgrammingIntro.pdf, 2006.
- Peter Dalgaard, "Introductory Statistics with R", 2nd edition, Springer, 2008.
- W. N. Venables, D. M. Smith, R Core Team, "An introduction to R", 2014. This book is freely available in http://www.r-project.org/ , option *Manuals* from *Documentation*.

Summary texts on R:

- Tom Short, "R Reference Card", http://cran.r-project.org/doc/contrib/Short-refcard.pdf, 2014

Web search on R:

- http://www.rseek.org/.