The Basics of R

Ahmad Abdel-Azim

December 23, 2018

Contents

About the R Language

- R is an interpreted language, not a compiled one, meaning that all commands typed on the keyboard are directly executed without requiring to build a complete program like in most computer languages (C, Fortran, Pascal,. . .).
- Intuitive syntax
- When R is running, *variables*, *data*, *functions*, *results*, etc, are stored in the active memory of the computer in the form of *objects* which have a *name*.
- The user can do actions on these objects with *operators* (arithmetic, logical, comparison, . . .) and *functions* (which are themselves objects).

Help Pages

- When unsure how to use a function or operator, look it up in the help pages (i.e. `?lm`, which will display the help page for the linear model function).
- **Description:** provides a brief description.
- **Usage**: for a function, gives the name with all its arguments and the possible options (with the corresponding default values); for an operator gives the typical use.
- **Arguments:** for a function, details each of its arguments.
- **Details:** detailed description.
- **Value:** if applicable, the type of object returned by the function or the operator.
- **See Also:** other help pages close or similar to the present one.
- **Examples:** some examples which can generally be executed without opening the help with the function `example`.

Data in R

Objects in R

- Objects in R have two *intrinsic attributes*: **mode** and **length**
- The mode is the basic type of the elements of the object.
- Four main modes: numeric, character, complex, and logical (FALSE or TRUE)
- The length is the number of elements of the object
- To display the mode, use `mode()` and to dsiplay the length, use `length()`

More on Mode. . .

- A value of mode character is input with double quotes “. It is possible to include a quote in the value if it follows a backslash.

```r
x <- "Ahmad has \"good\" grades"
cat(x)
```

```
## Ahmad has "good" grades
```

- The following table gives an overview of the type of objects representing data.

Table 1: Overview of the type of objects representing data.

Object	Modes	Several.modes.possible.
vector	numeric, character, complex, or logical	No
factor	numeric or character	No
array	numeric, character, complex, or logical	No
matrix	numeric, character, complex, or logical	No
data frame	numeric, character, complex, or logical	Yes
ts	numeric, character, complex, or logical	No
list	numeric, character, complex, logical, function, expression, …	Yes

Types of Objects

- **Vector:** a variable in the commonly admitted meaning
- **Factor:** a categorical variable
- **Array:** a table with *k* dimensions
- **Matrix:** a particular case of array with k = 2 (Note that the elements of an array or of a matrix are all of the same mode)
- **Data frame:** a table composed with one or several vectors and/or factors all of the same length but possibly of different modes
- **ts:** a time series data set, containing additional attributes such as frequency and dates - - **List:** can contain any type of object, including lists
- For some objects, information other than mode and length is necessary to describe the data and it is given by *non-intrinsic* attributes. Among these attributes, we can cite `dim()` which corresponds to the dimensions of an object.

Converting objects

- To convert an object from a type to another, changing some of its attributes, use a functions of the type `as.something` (i.e. `as.numeric()`).
- The following table summarizes common conversions…

Table 2: Object conversions.

Conversion.to	Function	Rules
numeric	as.numeric()	FALSE = 0, TRUE = 1
-	-	1, "2", . . . = 1, 2, . . .
-	-	A, . . . = NA
logical	as.logical()	0 = FALSE
-	-	other numbers = TRUE
-	-	FALSE, "F" = FALSE
-	-	TRUE, "T" = TRUE
-	-	other characters = NA
character	as.character()	1, 2, . . . = "1", "2", . . .
-	-	FALSE = "FALSE"
-	-	TRUE = "TRUE"
- There are func	tions to convert	the types of objects (`as.matrix`, `as.ts`, `as.data.frame`, `as.expression`,…). These

Working Directories in R

- The default behavior of R for the handling of .RData files and workspaces encourages and facilitates a model of breaking work contexts into distinct working directories.
- To change the working directory, use `setwd()` and include the folder path in the parantheses (in quotes).
- To check the current working directory use `getwd()` and include the folder path in the parantheses (in quotes).
- RStudio *projects* make it straightforward to divide work into multiple contexts, each with their own working directory, workspace, history, and source documents.
- To import data into R, use `read.csv()` or `read.table()` and include the file path (in quotes) inside the parentheses.

Generating Data in R

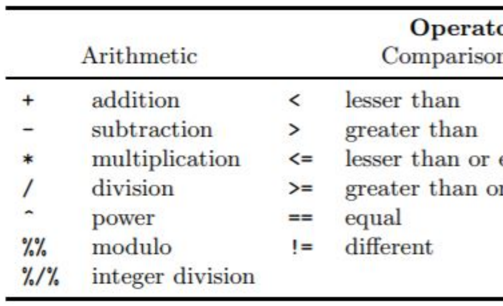
- **Regular sequence** (examples)..
- `1:30`
- `seq(length=9, from=1, to=5)`
- `c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)`
- `rep(1, 30)` or `gl(3, 5, length=30)`
- **Random sequences**: all functions are of the form `rfunc(n, p1, p2, ...)`, where *func* indicates the probability distribution, *n* the number of data generated, and *p1, p2,...* are the values of the parameters of the distribution (examples: `rnorm()`, `runif()`)
- Most of these functions have counterparts obtained by replacing the letter *r* with...
- *d* to get the probability density `dfunc(x, ...)`
- *p* to get the cumulative probability density `pfunc(x, ...)`
- *q* to get the value of quantile `qfunc(p, ...)`, with $0 < p < 1$.
- The table below gives the details for each distribution, and the possible default values...

Table 3: Types of distributions in R.

Law	Function
Gaussian (normal)	rnorm(n, mean=0, sd=1)
Exponential	rexp(n, rate=1)
Gamma	rgamma(n, shape, scale=1)
Poisson	rpois(n, lambda)
Weibull	rweibull(n, shape, scale=1)
Cauchy	rcauchy(n, location=0, scale=1)
Beta	rbeta(n, shape1, shape2)
'Student' (t)	rt(n, df)
Fisher-Snedecor (F)	rf(n, df1, df2)
Pearson (?2)	rchisq(n, df)
Binomial	rbinom(n, size, prob)
Multinomial	rmultinom(n, size, prob)
Geometric	rgeom(n, prob)

Law	Function
Hypergeometric	rhyper(nn, m, n, k)
Logistic	rlogis(n, location=0, scale=1)
Lognormal	rlnorm(n, meanlog=0, sdlog=1)
Negative binomial	rnbinom(n, size, prob)
Uniform	runif(n, min=0, max=1)
Wilcoxin's statistics	rwilcox(nn, m, n), rsignrank(nn, n)

Operators in R



		Operato	
Arithmetic		Compariso	
+	addition	<	lesser than
-	subtraction	>	greater than
*	multiplication	<=	lesser than or e
/	division	>=	greater than o
^	power	==	equal
%%	modulo	!=	different
%/%	integer division		

- There are three types of operators in R: *arithmetic*, *comparison*, and *logical*.
- The arithmetic and comparison operators act on two elements.
- The arithmetic operators act on variables of mode numeric, complex, or logical variables (logical values are coerced into numeric).
- The comparison operators may be applied to any mode and returns one or several logical values.
- The logical operators are applied to one or two objects of mode logical, and return one (or several) logical values.

Accessing the values of an object: the indexing system

- The indexing system is an efficient and flexible way to access selectively the elements of an object; it can be either numeric or logical.
- Use **brackets** to refer to and index.
- For example…

```
x <- 1:5
x
```

```
## [1] 1 2 3 4 5
```

```
x[3] <- 20
x
```

```
## [1]  1  2 20  4  5
```

Matrices

- For matrices, the indexing system can also be used; however, columns or rows must be specified.
- R has facilities for matrix computation and manipulation. The functions `rbind()` and `cbind()` matrices with respect to the lines or the columns, respectively.

Graphics in R

Managing Graphical Devices

- When a graphical function is executed, if no graphical device is open, R opens a graphical window and displays the graph. The list of available graphical devices depends on the operating system. One can open a graphical window with the command `x11()`.
- The list of available graphical devices can be found with `?device`.
- To change the active device, use `dev.set()` with the number of the active device in the parentheses.
- To close all graohical devices, use `dev.off()`.

Graphical Functions

- Common graphical functions in R include…
- `plot(x, y)`
- `boxplot(x)`
- `hist(x)`
- “
- In addition, symbols can be added to plots, using `symbols()` (see `?symbols`).
- For example… `hist(rnorm(n = 100, mean = 5, sd = 2))`
  Shiny applications not supported in static R Markdown documents

Low-level plotting commands

- R has a set of graphical functions which affect an already existing graph; they are called *low-level plotting commands*.
- Here are some common ones…
- `points(x, y)`
- `lines(x, y)`
- `text(x, y, labels,...)`
- `segments(x0, y0, x1, y1)`
- `abline(a, b)`
- `legend(x, y, legend)`

Graphical parameters

- The presentation of graphics in R can be improved with graphical parameters.
- They can be used either as options of graphic functions (but it does not work for all), or with the function `par()` to change permanently the graphical parameters.
- An exhaustive list of such garphical parameters are available on the par help page (`?par`).
- Common ones include: `col`, `lwd`, `lty`, `mfrow`, `pch`
- An example of plot using several graphical parameters…

```
honeyGlu.d1 <- read.csv("C:/Users/Ahmad Abdel-Azim/Documents/Research/Osmolality Exp/Version 1 May 30/h
par(mar = c(5,5,2,2))
Mconc <- honeyGlu.d1.zeros[honeyGlu.d1.zeros$Trt == "Manuka",]
plot(Mconc$Conc, Mconc$OD, ylim = c(), xlim = c(), pch = , cex = , col = , ylab = " ", xlab = " ", main
Mconc.line <- tapply(Mconc$OD, Mconc$Conc, mean)
lines(as.numeric(names(Mconc.line)), Mconc.line, col = , lwd = , lty = , lwd = )
```

Result…

Shiny applications not supported in static R Markdown documents

Statistical Analyses in R

Linear Models

- Basic structure: `lm(y ~ x + z)` (read as *Predict y from the additive effects of x and of z*)
- To account for the interactions between x and z, use `lm(y ~ x:z)`
- To account for the interactions between x and z as well as their additive effects, use `lm(y ~ x*z)` (identical to `lm (y ~ x + Z + x:z)`).
- To account for the quadratic effects, cubic effects,… nth-degree effects of a factor, use `lm(y ~ x + ploy(z, n))`.
- To account for the effects of b, which are nested in a, use `lm(y ~ x %in% z)` (identical to `x+x:z`, or `x/z`).
- To remove the effect of a factor, use `lm(y ~ x*z - x:z)` (identical to `x + z`).
- The functions that we can use subsequently to extract the results (from any class of statistical function output) are called *generic* functions.
- The table below summarizes some common generic functions…

Table 4: Generic Functions in R.

Function	Description
print	returns a brief summary
summary	returns a detailed summary
df.residual	returns the number of residual degrees of freedom
coef	returns the estimated coefficients (sometimes with their standard-errors)
residuals	returns the residuals
deviance	returns the deviance
fitted	returns the fitted values
logLik	computes the logarithm of the likelihood and the number of parameters
AIC	computes the Akaike information criterion or AIC (depends on logLik())

Supplementary Analyses

- Some generic functions do supplementary analyses from an object resulting from an analysis
- Usually that object is the main argument, but in some cases a further arguments are necessary.
- The `predict()` function can be used, for example, to compute the predicted values for new data from a fitted linear model.
- `Predict()` can be used in data visualization (i.e. to plot a linear model.)

Packages in R

- To install packages into R, use `install.packages(" ")` and type the name of the package into the paranthses (in quotes).
- To load this installed package, use `library()` and include the package name within the parentheses.
- To update all packages within R, use `update.packages()`.

Programming with R in Practice

Loops and vectorization

- Looping is simply automating a multi-step process by organizing sequences of actions or â€˜batchâ€™ processes and by grouping the parts that need to be repeated.
- Common types of loops: *for* loop and *while* loop.

For loops

- Format example…

```
# Create a vector filled with random normal values
u1 <- rnorm(30)
print("This loop calculates the square of the first 10 elements of vector u1")

# Initialize usq
usq <- 0

for(i in 1:10) {
  # i-th element of `u1` squared into `i`-th position of `usq`
  usq[i] <- u1[i]*u1[i]
  print(usq[i])
}

print(i)
```

- For loops can also be nested.

While loops

- The while loop is made of an initialization block, followed by a logical condition. This condition is typically expressed by the comparison between a control variable and a value, but any expression that evaluates to a logical value (True or False) is legitimate.
- If the result is FALSE, the loop is never executed.
- If the results is TRUE, the instruction or block of instructions is executed.
- Format example…

```
# Your User Defined Function
readinteger <- function(){
  n <- readline(prompt="Please, enter your ANSWER: ")
}

response <- as.integer(readinteger())

while (response != 42) {
  print("Sorry, the answer to whatever the question MUST be 42")
  response <- as.integer(readinteger())
}
```

- The `break` statement can be used to exit from a loop.
- The `next` statement discontinues a particular iteration and jumps to the next cycle.

Creating Functions in R

- Most of Râ€™s work is done with functions which arguments are given within parentheses. Users can write their own functions, and these will have exactly the same properties than other functions in R, allowing an efficient, flexible, and rational use of R.

- Format…

- R uses a rule called *lexical scoping* to decide whether an object is local to the function, or global.

- In simple terms, this just means that variables will only be accessed from the local environment if this variable is not already defined within the function.

Further Explanations

Link to pdf of *R for Beginners*.