Statistical Methods Fall 2021

Assignment 0: Introduction to *R*

Deadline: Friday November 5, 23:59

Here we present a short introduction to the statistical package *R* and a set of exercises, which will get you acquainted with the programming language *R*. It is recommended to use *R* via the interface *RStudio*[1].

Using *R* in *Rstudio*

Start *RStudio* by clicking on the *RStudio*-icon. This opens an *RStudio* environment with three windows: on the left a window called *Console*, top-right a window that shows the contents of your `Workspace` or your *R* `history`, and bottom-right a window of which the most important functions are to show the plots that you make and to display the Help items (select in the window's menu bar `Plots` and `Help`, respectively).

By selecting `Session -> Set Working Directory` you can choose a default directory (= folder in your Windows system). *R* will look in that directory for data files and save files to that directory.

In the *Console* window, *R* can be used interactively by typing *R* commands. The sign > at the beginning of a line is the *R*-prompt. It is followed by the commands that you type. If you type a command that is not syntactically complete and hit the *enter* key, then *R* will show a + to ask for more. You can type commands over several lines by grouping them within braces { and }.

You may prefer to work from the *Script* window. For this you choose `File -> New -> R Script` in the *RStudio* menu bar. This opens a fourth window. This *Script* window can be used as text editor in which a sequence of commands, or even a full program, can be written, and run. Contrary to the commands that you type in the *Console* window, the contents in the *Script* window can be saved (select the *Script* window and choose `File -> Save` in the *RStudio* menu bar), and reopened later (for this select `File -> Open`). In the script window you can have several R script files open simultaneously; the names of the open files are shown in the top row of the script window's menu bar.

The default file name extension for script files (suggested by *R* when you save a script) is `.R`, but script files are just plain text files, and can be edited also in your favorite text editor. You run parts of the text in a script window by selecting the text and typing <CONTROL-ENTER> or choosing `Run` from the script window's menu.

By typing <CONTROL-Q>, or with the command `q()` in the *Console* window, you can quit *RStudio*.

An introduction to R is available on Canvas. It is also available in *RStudio* (select `Help` in the lower-right window and then the option `An Introduction to R` in the track-down menu indicated by the black triangle).

Note. In each of the homework assignments, specific *R*-functions will be given which are useful for that assignment.

[1] If you want to use *R* on your own computer, you can download the *R* package from http://www.r-project.org. If you want to use *R* via *RStudio* on your own computer, you first need to download *R*. Next download *RStudio* from http://www.rstudio.com.

Exercises

Solve the following exercises in *RStudio* as efficiently as possible. Of these exercises hand-in: **only** the plots of Exercise 0.6 and the *R*-code of the functions that you wrote in Exercise 0.7 (maximum 2 pages in total). This assignment has to be handed in and there will only be a binary grading (pass/fail) which will not count toward your grade.

Exercise 0.1 *Examples*
The two tables below illustrate some of the possibilities in *R*. On the left-hand side are the *R*-commands; behind the # sign[2] a short description is given of what the command does.
Type the commands of the *first* table one by one in the *Console* window and study the results. After typing a command hit the *enter* key!

Table 1	**general**
`?hist`	# gives documentation about the function `hist`
	# (you can use `?` with the name of any other *R*-function
	# to find out what this function does and how to use it)
	vectors
`x1=c(1,5,-3)`	# creates a vector `x1` consisting of numbers 1, 5, -3
`x1`	# prints the elements of the vector `x1` on the screen
`x2=1:20`	# creates a vector `x2` with values $1, 2, \ldots, 20$
`y=c(x1,x2)`	# builds a vector `y` by concatenating `x1` and `x2`
`y[2]`	# extracts the 2nd element of a vector `y`
`y[1]=-5`	# replaces the 1st element of a vector `y` with -5
`u=seq(0,10,length=100)`	# creates a vector of 100 equally spaced numbers between 0 and 10
`round(u,1)`	# rounds u to 1 digit after the decimal point;
	# but be careful: sometimes .5 is rounded off!
`x=rep(x1,4)`	# creates a vector `x` by 4 times replicating the vector x1
`length(x)`	# outputs the length of the vector x
`s=sample(1:100,20)`	# draws a sample of size 20 from the numbers $1, \ldots, 100$
	arithmetic
$2 * 3 - 7;\ 2 \wedge 3$	# performs the two commands separated by ';'
`x+2`	# adds 2 to every element of x
`y=2*x`	# multiplies every element of x by 2 and saves the result in a vector `y`
`x+y`	# adds x and y element-wise
	data handling
`sum(x)`	# calculates the sum of the values in `x`
`mean(x)`	# calculates the mean of the values in `x`
`sd(x)`	# calculates the standard deviation of the values in `x`
`var(x)`	# calculates the variance of the values in `x`
`sort(x)`	# sorts the elements of `x` from small to large
`x[x<0]`	# selects the negative elements in `x`
`sum(x<0)`	# counts the number of negative elements in `x`
	reading data from file
`z=scan("C:/data.txt")`	# reads data from the file data.txt that was saved
	# in the folder C: and saves them in *R* in a vector `z`
`z=scan("")`	# asks for input interactively: copied data can be pasted now,
	# and will be saved in a vector `z` after typing the *enter* key
`source("commands.txt")`	# reads the file commands.txt in your current
	# working directory and runs the entire file as R-code
	# also works for .R-files

[2] In *R* everything that follows after a # sign is ignored and does not influence the computations.

Exercise 0.2 *Examples (continued)*
The commands given in Table 2 are saved in the script file `intro-table2.R`. Download this file from Canvas into one of your own folders, and open the file in *RStudio*. Again, run the commands one by one, but now in the Script window, and study the results.

Table 2	**matrices**
`m=matrix(x,nrow=3,ncol=4)`	# creates a 3 by 4 matrix `m` from vector `x` column-wise
`m=matrix(x,nrow=3,ncol=4,byrow=T)`	# creates a 3 by 4 matrix `m` from vector `x` row-wise
`m[2,3]`	# extracts element on 2nd row and in 3rd column of `m`
`m[3,]`	# extracts 3rd row of `m`
`colSums(m)`	# calculates column sums of matrix `m`
`apply(m,1,max)`	# gives maximum of each row of `m`
	# (replacing 1 by 2 does it for columns, and
	# `max` can be replaced by some other *R*-function)
`cbind(x,y)`	# forms a matrix with columns x and y
`M=read.table("data.txt")`	# reads values from file and saves in a table M
`M=as.matrix(M)`	# converts table M into a matrix M
lists	
`L=list(1:2,4:6)`	# creates a list that consists of two elements
`L[[1]]`	# extracts the first element of the list `L`
`L=list(element1=1:2,x)`	# creates a list that consists of two elements also
`L$element1`	# extracts the first element of the list `L`,
	# but now the elements of the list must have names
for-loops	
`v=numeric(0)`	# creates a numeric object v of length zero
`for(i in 1:100) {`	# does 100 times:
`v[i]=mean(sample(1:100,25)) }`	# take a sample of size 25 out of $1, \ldots, 100$ and
	# compute sample mean and store in v
plotting	
`hist(v)`	# plots a (unscaled) histogram of v
`hist(v,prob=T)`	# plots a scaled histogram of the values in `v`
`u <- v^2`	# takes the square of each entry in `v` and saves the resulting vector in `u`
`plot(u,v)`	# plots the points `(u[i],v[i])` in the plane
`plot(u,v, xlab="u",ylab="v",`	# also names the axes,
`main="Plot of u against v")`	# and gives the plot a title
`points(u,v+5,col="blue")`	# adds the points `(u[i],v[i]+5)` in blue
`plot(u,v,type="l",`	# plots line segments between points `(u[i],v[i])`,
`col="red", lty=2)`	# with dashed, red lines
`plot(sort(u),sort(v),type="l",`	# plots line segments between points with sorted coordinates
`col="green", lwd=3)`	# with thick, green lines
creating a function	
`add2=function(x){`	# creates a function `add2` with one argument x,
`x+2}`	# which adds 2 to every element of x
`add2(c(3,5))`	# calculates function `add2` for values 3 and 5
`fibonacci=function(n){`	# creates a function `fibonacci` with argument n, which
`w=numeric(n)`	# creates a vector w of length n consisting of zeros,
`w[2]=1`	# assigns value 1 to the second element of `w`,
`for(i in 3:n) {`	# uses for-loop to create new elements up to n-th element
`w[i]=w[i-2]+w[i-1] }`	# out of two existing elements,
`w }`	# and outputs w
`fibonacci(20)`	# computes first 20 Fibonacci numbers

Note 1. It is necessary that a vector has been created (with the *R*-function `numeric`) before assigning values to or performing operations on specific elements of it in a for-loop.
Note 2. When you create a function in which the desired output is stored in an *R*-object, put the name of this *R*-object as the last line of the function. Otherwise, the function does not give output.

Exercise 0.3 *Vectors*

a. Create a vector x consisting of the numbers $20, 0.16, 36$ and -66.4.

b. Add to this vector the number 58 on the fifth position.

c. Order the elements of x from small to large and call the vector of ordered numbers y.

d. Multiply each element of x by 1.31. Assign the vector of multiplied numbers to x again.

e. Round each element of the new x to one decimal place.

f. Select all positive elements of the new x.

g. Change the third element of the new x into 12.

Exercise 0.4 *Matrices*

a. Create a vector $x2$ consisting of the sequence of numbers between 5 and 13 with step size 0.3. Create a matrix m with nine rows and three columns in which the 27 numbers in $x2$ are ordered column wise.

b. Extract the element on the fifth row and in the first column of m.

c. Extract the second column of m.

d. Compute the mean value of the numbers in m.

e. Compute the mean value of each column of m by using the *R*-function `apply`.

Exercise 0.5 *Data*
Data can come in several ways. They can come as values saved in a file, as a file containing *R*-code for creating a data object (vector, matrix, list) in *R*, or saved as *R*-data objects in a working directory. Download from Canvas the files `lepton` and `sample1`, and the workspace file `Ass0.RData` into one of your own folders.

a. The file `lepton` in Canvas contains the *R*-code for creating in *R* the list `lepton` with 5 components. Create this list. You could use `source()` for this. Next, extract the fourth component of the list. Finally, extract (with a single command) the second element of the first component of the list.

b. Use the *R*-function `scan` to create a vector with the data in the file `sample1`. Transform this vector into a matrix with four rows and five columns. Then use the *R*-function `dimnames`, to assign to the rows and columns of this matrix the names a, b, c, d, and I, II, III, IV, V, respectively.
Note. You can also use `as.matrix(read.table(...))` to read in these data as a matrix in one go.

c. Open the workspace file `Ass0.RData` in the *Workspace* window. It contains an object `school`. Now type the name of this object in the *Script* or the *Console* window to see what it contains.

Exercise 0.6 *Plots*
Put the three plots of the parts a, b and c of this exercise next to each other in one picture with the command `par(mfrow=c(1,3))`. This command should be given once, before giving the commands for making the pictures. The plot of the part d should be plotted separately. To get only one plot in the plot window again, you can override the previous command by typing `par(mfrow=c(1,1))`.
Note. You can use the function `Zoom` in the *Plots* window and re-scale the picture to view it with appropriate dimensions.

a. Plot a scaled histogram of the data in `sample1`. Give the histogram an appropriate title.

b. Draw the function $y = x^4$ for $x \in [0, 2]$. Give the picture an appropriate title. Add the line $y = 5x + 2$.
Note. For drawing smooth functions the vector plotted on the x-axis needs to contain a sufficiently dense grid of points.

c. Investigate what the three commands `is.numeric(school)`, `table(school)` and `barplot(table(school),col=c("blue","green", "black","yellow"))` do.

d. Study the help documentation of the *R*-function `pie`. Make a pie chart with your favourite colors, for pie sales proportions $0.37, 0.17, 0.28, 0.07, 0.06, 0.05$, for the pie flavours "Cheesecake", "Pear", "Chocolate", "Almond", "Strawberry", and "Other", respectively. Give the pie chart an appropriate title.

Exercise 0.7 *Writing your own functions*
Write and test the following functions.

a. A function that subtracts 3 from every element of a vector x, derives the element-wise square of the resulting vector, and that has the result as its output.

A vase contains 30 red and 70 white balls. For b and c use the *R*-function `sample`.

b. A function that draws, with replacement, 50 balls from the vase, that also computes the fractions of red and white balls in the draw, and that has these fractions as its output.

c. Rewrite the function of part b for the situation that n balls are being drawn with replacement from a vase with mr red balls and mw white balls. Take as arguments n, mr and mw with default values $n = 10$, $mr = 30$ and $mw = 70$.

d. A function that
– creates a vector u of length 200;
– does 200 times:

- draw a sample of size 40 from 1,. . .,90,
- calculate the median of the sample,
- save the median as an appropriate element of u;

– outputs u.
Hint: use a for-loop.