Practical Session on R

Louis-Claude Canon and Emmanuel Jeannot

April 2010

Abstract

The objective of this practical session is to present some basic features provided by R. Additionally, methodological principles are given in order to acquire autonomy in using R functions. The participant is assumed to be familiar with programming, file editing and Unix shell.

1 Getting used to R interactive environment

If R is not installed on your machine, you can download and install it from the R project website.

R is also available on Grid5000 machines. You may use your account to reserve one machine.

Then, launch R:

$ R

After some information, the issued prompt is:

>

You can now enter your first assignment:

> X <- c(5, 1)

Type the name of the variable X to show its content:

> X

[1] 5 1

Or, you can do the assignment and show the content of the variable by adding parenthesis around the assignment:

> (X <- c(5, 1))

[1] 5 1

You can use an external file in which you will enter your R instructions. To load the script, use:
> source("script.r")

Note that parenthesis assignments does not print anything with scripts. Use instead the function `print`. See its manual page:

> ?print

Additional documentations can be found on the R project website.

Test the following functions without argument: `ls`, `rm`. You can then see the output of the following functions on any variable (try with the variable `X`): `mode`, `class` and `length`.

At any step, you can access to the previous instructions you have entered by using the `up` key or by exiting, saving your session and opening the file named `.Rhistory`.

## 2 Data types

In this section, you will be asked to perform simple operations on data types. Save relevant R instructions in a script for future usage during the session.

**Q1:** First, create a vector of 100 integers randomly drawn from a uniform distribution between 10 and 30. We refer you to the manual pages of the following functions: `runif` and `as.integer`.

**Q2:** Test if each integer in the range [10, 30] appears at least once (`length` and `unique` functions).

**Q3:** Count how much of them are greater than 15 (`which` function).

**Q4:** Copy this vector and add a Gaussian noise with variance 1 (`rnorm` function). You have now two vectors of distinct types (check with the `class` function).

**Q5:** Build a bi-dimensional structure with both vectors (`cbind` for generating a matrix or `data.frame`). Test these two methods and use `attributes` to witness the difference. Only one of these methods preserves the type of each vector. Which one?

**Q6:** Give an appropriate name to each column either at the construction of the structure (when calling `data.frame` for example) or by modifying the attributes (`names` or `colnames`). You can now access one column with the following syntax: `data$colname`.

**Q7:** Now, we discard the rows in which the second value (the numeric one with the noise) is outside the range [10, 30]. Hints: vectorized or `|` and `which`. Note that `T[1:2,]` returns the two first rows.

**Q8:** Count how much rows are left (`dim`).

**Q9:** Suppose we want to keep all these values (in total, we have four vectors). The lengths of these vectors are different. Thus, we will put them in a list with two values: the two initial columns and the result after discarding some values (`list`).

You can access to the field of the list using the syntax: `data[[index]]`.

**Q10:** Lastly, build a function that takes as parameters the number of integers that will be generated and the range. The result is the list described previously.

**Q11:** If no value are discarded, we would like to return a data frame instead of a list. Position a condition before the end of the function.

**Q12:** Reimplement the function `sum` using a loop `for`. Then, compare the performance of both versions.
3 Inputs

We would like to save in a file the first data frame that was created.

Q1: Use the function `write.table` (the first argument in the data frame and the second is the name of the output file) to perform this operation and open the generated file to note the default behavior.

Q2: Use `read.table` on the same file with the default parameters (precise only the name of the file) and compare the content and the attributes with the initial data frame.

Here are the specifications of a simple CSV file: the separation between each value on the same row must be a comma, the row and column names must not appears.

Q3: Repeat the previous write and read operations while storing data into such a CSV file.

4 Plot

In R, it is possible to specify the device driver to use for each graphical output. By default, it is the one of your graphical environment. You can generate postscript, pdf, xfig, png, ... Refer to the manual page of `Devices` for more details.

If you are using a Grid5000 machine, you may use the pdf device driver for each figure. Before calling a plotting primitive, initialize the device with `pdf("test.pdf")` and close the device afterward with `dev.off()`.

4.1 `plot` function

The first notable graphical function, `plot`, allows several types of plot to be drawn: points, lines, both, steps, ...

Q1: We would like to plot the value used in the previous sections with the first column on the x-axis and the second column on the y-axis. Experiment various types of plot. When plotting lines, the values should be sorted in ascending order (you may already have seen what it looks like otherwise). The function `order` allows you to keep the same data structure without altering it.

Q2: Most of the parameters are described on the manual page of `par`. For instance, change the points aspect with parameters `cex`, `col` and `pch`. Lastly, experiment logarithmic scale (`log`).

Q3: Specify the title, x-axis and y-axis labels (with parameters `main`, `xlab` and `ylab`). Add a legend (`legend` function).

Q4: We will superpose another plot on this one. After the first plot, enter `par(new = TRUE)`. This specifies that the current window is new and does not need to be cleared. Be careful not to draw again the axis on the next plot as they will overlap with those of the first plot (`xaxt`, `yaxt` and `ann` parameters). The values to be used for this second plot are the same for the x-axis but the opposite ones for the y-axis. Then, the last axis on the right must be drawn (`axis` function).

4.2 Plotting distributions

In this section, we suppose that we have a collections of values and we want to plot them without excessive aggregation. The data is the second column of the data frame previously
used.

**Q1:** Initialize a plot by calling the function `hist`. Specify to use density scale (`freq = FALSE`). The breaks can be adjusted with the parameter `breaks` (10 to 20 breaks should be appropriate).

Low-level plotting does not initialize a plot but draw additional elements on existing plot (then, they use the existing axes).

Enter `lines(density(VALUEs, bw = 0.5))`, where `VALUEs` is your collection of values. This draws an estimate of the density of the distribution from which comes the values. Parameter `bw` denotes the sensitivity of the method to each single sample (try larger and smaller values).

**Q2:** All values can be drawn with the function `rug`.

**Q3:** Finally, R allows to plot empirical cumulative distribution functions. As the y-axis differs from the previous plot, we need to call a new high-level plotting function. Let us start with `par(new = TRUE)`. Then, you have to call `plot(ecdf(VALUEs))` (or the `plot.ecdf` function). The same tricks as before must be done for avoiding axis superposition. Note that calling `lines(ecdf(VALUEs))` would have worked if the axis were compatible. Try to put the parameter `do.points` to `FALSE` for a more concise graphic. You may have notice that the x-axis of this new plot differs slightly from the one initialized for the histogram. To force the range of the x-axis, you must give the same value to the parameter `xlim` in the high-level functions.

Copy and paste the example given on the manual page of function `ecdf`. It shows what can be obtained through customization.

### 4.3 Summarizing data

We assume that the previous approach for describing the statistical dispersion of the data has been applied. Moreover, we assume that the approach has revealed that the samples (or each set of samples) have a unique mode and that it is pertinent to summarize them. However, data may be inconsistent (several modes, no central tendency for instance) and the following functions might be irrelevant.

Using the first data frame produced (with integer and noise), we group the rows by their values in the first column. Then, we want to characterize the dispersion of all the values in the second column that have the same value in the first column. We will use boxplots. Use the following lines to generate the input of function `boxplot` (where `V` is the data frame):

```r
L <- sort(unique(V[,1]))
R <- sapply(L, function(x) return(V[V[,1] == x,2]))
names(R) <- L
```

Variable `L` contains the distinct integers generated randomly in the first part of the session. The function `sort` is more concise to use in this case than the function `order` that was already mentioned for sorting your points. The function specified in the call to `sapply` is executed for each of these integers. This function returns the values in the second column that corresponds to the same value `x` in the first column. As a data frame is a list that can be manipulated as a matrix, `V[[1]]` (first element of the list) is equivalent to `V[,1]` (first column of the matrix).

After that, a simple call to `boxplot(R)` draw the box-and-whisker plots. However, this procedure may be shortened by using the formula feature of R. Indeed, calling `boxplot(V[,2] ~ V[,1])` is equivalent (grouping values in vector `V[,2]`, by their values in `V[,1]`).
Q1: Now, the idea consists in superposing the points and the boxplots. You may begin with a plot call (with type = "p"). Then, use the parameter add and at of the boxplot function. There is no native function for plotting error bars with R. A function is available in the Hmisc package (loadable with function library) and there is a lot of useful packages on Crantastic.

Q2: Our solution is to draw the points and then to add manually each error bar with the function arrows (with an angle equals to 90°). In our case, the error bar must be centered on the mean and its length must be twice the standard deviation.

These error bars must not be used if they does not convey the same information as boxplots (for instance for asymmetric distribution).

Q3: Finally, you may use the function lines on your data if no significant deviation is observed.

4.4 Scatter plot

As you may have experienced it with the examples concerning the function ecdf, it is possible to draw several figures on the same plot in a matrix-like way.

There is two methods to generate such plots: manually or with the function pairs.

The manual method first step redefines the graphic settings:

\[
\text{op <- par(mfrow=c(nr, nc))}
\]

It stores the old setting in the variable op and specifies that figures will be arranged in a 'nr'-by-'nc' array. Each time a high-level graphic function is called, a new space is filled in the layout. At the end, initial graphic settings can be restored using \text{par(op)}.

Place the previous generated plots in a 2-by-2 array (first general plots, histogram and ecdf, boxplot and points, and lines).

In the cases when a data frame has more than 2 columns, the function pairs may be used to plot each column against each other column. You can test the first example given on the related manual page and compare with the data given in input.

5 Statistical test

This section show how to use some native statistical tests (normality and correlation).

5.1 Normality test

Normality tests can be used to validate a normality assumption about a set of random points.

Q1: Generate two vectors of 10 values that follow either a normal distribution (rnorm) or a uniform one (runif).

Q2: Call the Shapiro-Wilk test on both vectors. Find the correct function by searching the manual pages (using two interrogation points, ",?", followed by a string). Refer to the p-value for this test (rule of dumb: if greater than one then the sample "is" normally distributed).

Q3: Find the function for the Anderson-Darling test. It is included in a package that need to be loaded (use the function library). Repeat the previous steps. In addition to the p-value that should be high for normal samples, the statistic A is also close to zero.
5.2 Correlation test

Correlation gives an idea of the linear dependence that ties two samples of values. For instance, if a sample is exactly a linear function of another sample, then the correlation coefficient is equals to one. On the contrary, if there is no linear dependence, then the correlation coefficient is null.

Q1: Find the function that corresponds to this test by searching the manual pages. Many functions are returned when searching for the string "correlation". Most of them belongs to packages that need to be loaded. By default, however, the functions of the package stats are directly usable and one of them is the function that we are looking for.

Q2: Generate two samples of 1000 random values and measure their correlation.

Q3: Generate a sample of random values and produce another sample by a linear transformation from the first one. Measure the correlation between both dependent samples.

Q4: Repeat this step using a quadratic relation instead. The basic function measures the Pearson correlation coefficient. Use either the Kendall of Spearman method.

5.3 Computing confidence interval

We will now use the distribution functions for computing confidence interval. We are interested in an estimate that follows a normalized Student T distribution with 5 degrees of freedom. We would like to compute its confidence interval with a confidence level of 95%.

Q1: Find the manual page corresponding to the Student T distribution (either with "student" or "distribution" keywords, or with automatic completion when entering the letter "q").

Q2: The manual page describes four functions. We want to use the quantile function in order to get the value below which a sample is generated in 2.5% of cases. Similarly, obtain the upper bound (value beyond which a sample is generated in 2.5% of cases.

The confidence interval in the difference between the two obtained values.
6 Correction

6.1 Data types

Q1: \( X \leftarrow \text{as.integer(runif(100, min = 10, max = 31))} \)
Q2: \( \text{length(unique}(X)) \)
Q3: \( \text{length(which}(X > 15)) \)
Q4: \( Y \leftarrow X + \text{rnorm(length}(X)) \)
Q5: \( V \leftarrow \text{data.frame}(X, Y) \)

*cbind* builds an array (same type for each column). *data.frame* builds can have distinct types for each column.

Q6: \( V \leftarrow \text{data.frame}(\text{int}=X, \text{noise}=Y) \)
Q7: \( W \leftarrow V[V[,2] >= 10 & V[,2] <= 30,] \)
Q8: \( \text{nrow}(W) \)
Q9: \( Z \leftarrow \text{list}(V, W) \)
Q10: \( \text{generate} \leftarrow \text{function}(n, \text{range}) \{ \)
\( \quad X \leftarrow \text{as.integer(runif}(n, \text{min} = \text{range}[1], \text{max} = \text{range}[2]+1)) \)
\( \quad Y \leftarrow X + \text{rnorm(length}(X)) \)
\( \quad V \leftarrow \text{data.frame}(\text{int}=X, \text{noise}=Y) \)
\( \quad W \leftarrow V[V[,2] >= \text{range}[1] & V[,2] <= \text{range}[2],] \)
\( \quad \text{return(list}(V, W)) \)
\( \}
Q11: \( \text{generate} \leftarrow \text{function}(n, \text{range}) \{ \)
\( \quad X \leftarrow \text{as.integer(runif}(n, \text{min} = \text{range}[1], \text{max} = \text{range}[2]+1)) \)
\( \quad Y \leftarrow X + \text{rnorm(length}(X)) \)
\( \quad V \leftarrow \text{data.frame}(\text{int}=X, \text{noise}=Y) \)
\( \quad W \leftarrow V[V[,2] >= \text{range}[1] & V[,2] <= \text{range}[2],] \)
\( \quad \text{if} (\text{nrow}(V) == \text{nrow}(W)) \)
\( \quad \quad \text{return(cbind}(V[[1]], V[[2]])) \)
\( \quad \text{else} \)
\( \quad \quad \text{return(list}(V, W)) \)
\( \}
Q12: \( \text{ssum} \leftarrow \text{function}(x) \{ r \leftarrow 0; \text{for} (i \text{ in} x) r \leftarrow r + i; \text{return}(r) \} \)
\( \text{ssum(seq}(0, 1, 0.0000001)) \)
\( \text{ssum(seq}(0, 1, 0.0000001)) \)

6.2 Inputs

Q1: \( \text{write.table}(V, \text{"test.txt"}) \)
Q2: \( \text{read.table}("test.txt") \)
Q3: \( \text{write.table}(V, \text{"test.csv"}, \text{sep} = ",", \text{row.names} = \text{FALSE}, \text{col.names} = \text{FALSE}) \)
\( \text{read.table}("test.csv", \text{sep} = ",",) \)

6.3 Plot

6.3.1 *plot* function

Q1: \( \text{plot}(V[\text{order}(V[,1], V[,2]),], \text{type} = \text{"b"}) \)

7
Q2: plot(V[order(V[,1], V[,2]),], type = "b", cex = 0.5,
        col = "red", pch = 12)
Q3: plot(V[order(V[,1], V[,2]),], type = "b", cex = 0.5,
        col = "red", pch = 12, main = "Main title",
        xlab = "integer", ylab = "integer with noise")
legend("bottomright", "random value", pch = 12, col = "red", pt.cex = 0.5)
Q4: par(new = TRUE)
plot(V[,1], -V[,2], xaxt = "n", yaxt = "n", ann = FALSE)
axis(4)

6.3.2 Plotting distributions

Q1: hist(Y, freq = FALSE, breaks = 20)
Q2: rug(Y)
Q3: hist(Y, freq = FALSE, breaks = 20, xlim = range(Y))
    lines(density(Y))
    rug(Y)
    par(new = TRUE)
    plot(ecdf(Y), do.points = FALSE, xlim = range(Y),
         xaxt = "n", yaxt = "n", ann = FALSE)

6.3.3 Summarizing data

Q1: plot(V, type = "p")
    boxplot(V[,2] ~ V[,1], add = TRUE, at = L,
            xaxt = "n", yaxt = "n", ann = FALSE, bty = "n")
Q2: points(L, sapply(R, mean), cex = 2)
    arrows(L, sapply(R, mean), L, sapply(R, mean) + sqrt(sapply(R, var)),
           cex = 2, angle = 90, length = 0.05)
    arrows(L, sapply(R, mean), L, sapply(R, mean) - sqrt(sapply(R, var)),
           cex = 2, angle = 90, length = 0.05)
Q3: lines(L, sapply(R, mean))

6.4 Statistical test

6.4.1 Normality test

Q1: U <- runif(100)
    N <- rnorm(100)
Q2: ??"Shapiro-Wilk"
    shapiro.test(U)
Q3: library(nortest)
    ad.test(U)
6.4.2 Correlation test

Q1: ?correlation
   \texttt{?cor.test}
   \texttt{Q2: U1 <- runif(1000)}
   \texttt{U2 <- runif(1000)}
   \texttt{cor.test(U1, U2)}
   \texttt{Q3: U2 <- 10 \times U1 - 50}
   \texttt{cor.test(U1, U2)}
   \texttt{Q4: U2 <- U1^2}
   \texttt{cor.test(U1, U2)}

6.4.3 Computing confidence interval

Q1: \texttt{?stats::TDist}
   \texttt{Q2: qt(0.025, df = 5)}
   \texttt{qt(0.975, df = 5)}