Warm up session : R for beginners
– Level 1 –

1 Rudiments

For unitary commands, you can work directly in the console (after the automatic symbol >). As soon as you deal with a list of commands, it is better to create a script file.

1.1 Help

Each time you want to use a function, it is always useful to ask for any help on it (syntax, parameters, default values, outputs, etc.). You may also need an example of running.

```
> help(mean)
> ?mean
> example(mean)
```

1.2 Comments

Do not hesitate to comment your script (using symbol #), especially when it becomes complicated.

```
# Now I apply Einstein's relativity to compute the curvature of my space-time
c <- 1 # Well, maybe I'm wrong...
```

1.3 A calculator

Try to use R as a calculator, all basic mathematical symbols exist.

```
> (1+2)*(5-3)
> (5/2)^3
> sqrt(25)
> abs(-5)
> cos(pi/3) # Degrees or Radians ?
> 19%%3 # What does it mean ?
```

1.4 Variables and their types

Use an arrow to assign a value to a variable. Look at the variable's content directly by its name or through the `print` function.

```
> a <- 2
> print(a)
> b <- -3.2
> b
```

Standard types are automatically handled : integer, float, complex, boolean, string, etc. Use `class` to check the type of a variable. Belonging to the same class provides access to the same functions (just like object-oriented programming).

```
> exInt <- 2
> class(exInt)
> exFlo <- pi
> class(exFlo) # Both floats and integers are numeric
> exCom <- 1+1i
> class(exCom) # But complex numbers are not numeric...
> exBoo <- TRUE
> class(exBoo)
> exStr <- "hello"
> class(exStr)
```

Try to manipulate and combine them, using the comparison operators and their specific functions.

```
> deg360 <- exInt*exFlo
> Mod(exCom) == sqrt(2)
> deg360 < pi/2
> exCom - Re(exCom) - 1i*Im(exCom)
> !exBoo
> paste(exStr, "everybody", sep=" ")
> exStr + exInt # Hmm, what was I thinking ?
```

`R` has the advantage (or the defect) to proceed to mathematically questionable operations. Special characters exist to handle such results, do not hesitate to check your calculations.

```
> bigValue <- 1/0
> is.finite(bigValue)
> is.infinite(bigValue)
> undefValue <- 0/0
> is.nan(undefValue) # What is NaN ?
> sqrt(-1)
> is.nan(1i)
> 1i == sqrt(-1) # What is NA ?
> 0+1i == sqrt(-1+0i)
```

1.4.1 Vectors

A vector in `R` is treated as a column of values. Some shortcut functions exist to deal with values having a logical progression, and usual operations and comparisons of vectors are available.

```
> V1 <- c(1, 2, 5, -1, 2)
> V2 <- 1:5
> c(V1, V2)
> V1*V2 # Is it a scalar product ?
> length(V1)
> V1[3]
> V2[10] # Why ?
> V3 <- seq(-5, 5, by=2)
> 10:0
> t(10:0) # What's the difference ?
> V4 <- rep(1, 6)
```

```
> V3 >= V4
> V3%*%V4
> which(V3 < 2)
> sort(V3)
> sort(V3, decreasing=TRUE)
```

Sometimes we may be required to change dynamically the length of a vector, because we have no prior information on the amount of data to be stored.

```
> EmptyVec <- c() # Empty vector
> length(EmptyVec)
> EmptyVec <- c(EmptyVec, 0)
> length(EmptyVec) # Note that () =/= (0)
> EmptyVec <- c(EmptyVec, 1)
> EmptyVec <- EmptyVec[-1]
> length(EmptyVec)
```

1.4.2 Matrices

We create a matrix from a vector, specifying the number of rows or columns needed.

```
> M <- matrix(c(2, 3, 5, 7, 11, 13), ncol=2)
> M
> dim(M)
> nrow(M)
> ncol(M)
> N <- matrix(c(2, 3, 5, 7, 11, 13), ncol=2, byrow=TRUE)
> N
> Z5 <- matrix(0, nrow=5, ncol=5)
> I5 <- diag(5)
> diag(1:5)
> diag(I5) # What's the difference ?
> M[1,2]
> M[3,4] # Why ?
> M[3,]
> M[2:3,2]
> M[-2,]
```

We add rows or columns using `rbind` and `cbind`. As for vectors, it enables to change dynamically the dimensions of the matrix.

```
> rbind(M, N)
> cbind(M, N)
```

Like vectors, usual operations and comparisons of matrices are available. As it is shown in the examples below, they have to be carefully used.

```
> M+N
> M-N
> M/N # What is this strange division between matrices ?
> M*N
> t(M)%*%N # What's the difference ?
> M^2 # Is it a matrix product ?
> A <- matrix(c(1, 3, 2, -4), nrow=2)
> eigen(A) # How to access to values and vectors separately ?
```

```
> det(A)
> solve(A) # Why ‘solve’ to inverse ?
> A == 1
```

1.4.3 Lists

A list is a generic vector that may contain different objects having a label.

```
> V <- c(158, 124, 182)
> a <- 22
> s <- 1.85
> n <- "Jon Snow"
> Indiv <- list(Name = n, Size = s, Age = a, KilledEnemies = V, isAStark = TRUE)
> Indiv
> Indiv[[1]]
> Indiv$Age
> Indiv[[4]][1] <- Indiv[[4]][1]+1 # The fourth element of the list is a vector
> Indiv$KilledEnemies
> summary(Indiv)
```

1.4.4 Dataframes

A dataframe is a generic matrix that may contain different types of rows or columns, having a label.

```
> DF <- data.frame(C1 = 1, C2 = 1:10, C3 = letters[1:10])
> DF
> colnames(DF)
> dim(DF)
> DF[3:5,]
> DF[-10,]
> rbind(DF, c(1, 1, "a"))
> DF # Why didn’t it change ?
> DF <- cbind(DF, 10:1)
> colnames(DF)[4] <- "C2inv"
> rownames(DF) <- paste("R", 1:10, sep="")
> DF["R3","C2inv"]
```

1.5 Descriptive statistics

For numeric vectors, descriptive statistics are easily handled with the numerous associated functions.

```
> n <- 1000
> X <- rnorm(n, mean=3, sd=2)
> m <- mean(X)
> var(X)
> sum((X-m)^2)/n # The difference ?
> sum((X-m)^2)/(n-1)
> median(X)
> quantile(X)
> quantile(X, probs=c(0.3, 0.6, 0.9))
> min(X)
> max(X)
```

2 Conditions and Loops

Like the usual programming languages, `R` is able to deal with conditions and loops. Note that we use `==` to test for equality whereas we use `!=` to test for difference and `<`, `<=`, `>`, `>=` to test for comparisons.

2.1 Logical operators

We first give the syntax associated with the standard logical operators.

2.1.1 Or

The syntax is `A | B` to test for "`A` or `B`".

2.1.2 And

The syntax is `A & B` to test for "`A` and `B`".

2.1.3 Not

The syntax is `!A` to test for "not `A`".

2.1.4 Xor

The syntax is `xor(A, B)` to test for "`A` xor `B`".

```
> a <- 1
> b <- 2
> (a == 1) # Essential, crucial : see the difference between 'a = 1' and 'a == 1'
> (b == 1)
> (a != 1)
> (a == 1) | (b == 1)
> (a == 1) & (b == 1)
> !(b == 1)
> (b != 1) == !(b == 1) # What ??
> (a == 1) | (b == 2)
> xor((a == 1), (b == 2)) # What's the difference between 'or' and 'xor' ?
```

2.2 Block if-else

The syntax is `if (cond)` { `instr` } `else` { `instr` } where the `else` block is optional. An `ifelse` shorcut is also available.

```
# Let's flip a coin
x <- runif(1)
if (x < 0.5) {
    print("Heads")
} else {
    print("Tails")
}

ifelse(runif(1) < 0.5, "Heads", "Tails")
```

2.3 Loop for

The syntax is `for (var in seq) { instr }`.

```
# Let's enumerate the alphabet
for (i in 1:length(letters)) {
    print(letters[i])
}
```

Note that the sequence is not necessarily numeric, for example we can look through a list.

```
# What are the registered properties of Indiv ?
for (prop in Indiv) {
    print(prop)
}
```

2.4 Loop while

The syntax is `while (cond) { instr }`.

```
# Let's compute the sum of the first n terms of a geometric sequence
q <- 1/3
n <- 20
s <- 0
i <- 0
while (i <= (n-1)) {
    s <- s+q^i
    i <- i+1
}
print(paste("Sum :", s))
print((1-q^n)/(1-q)) # Faster ?
```

2.5 Loop repeat

The syntax is `repeat { instr if (cond) { break } }`.

```
# Let's compute the terms of an arithmetic sequence until it exceed N
r <- 1/3
N <- 100
s <- 0
i <- 0
repeat {
    s <- s + r
    i <- i+1
    if (s > N) {
        break
    }
}
print(paste("Index :", i))
print(paste("Value :", s))
```

3 Functions

We can also define our own functions. The syntax is `name = function(arg) { instr return(var) }`, where the `return` command is optional. Some examples are provided below.

3.1 No output

If your function does not need to return any value, then do not use the `return` command.

```
# Flip n coins with heads probability p
flipcoins <- function(n, p) {
    for (i in 1:n) {
        x <- runif(1)
        if (x < p) {
            print("Heads")
        } else {
            print("Tails")
        }
    }
}

flipcoins(10, 0.1)
flipcoins(15, 0.5)
flipcoins(2, 0.9)
```

3.2 Unique output

Use `return(val)` to return the result of a treatment in your function.

```
# Concatenate 3 vectors into a single matrix
concat <- function(V1, V2, V3) {
    Mat <- cbind(V1, V2, V3)
    return(Mat)
}

M <- concat(c(1,0,0), c(0,1,0), c(0,0,1))
M <- concat(rnorm(10), runif(10), rbinom(10,5,0.2))
```

3.3 More than one output

A simple method to produce more than one output is to create a list with all required variables.

```
# Estimate mean and variance of a sample
estimMV <- function(Sample) {
    m <- mean(Sample)
    v <- var(Sample)
    out <- list(Mean = m, Var = v)
    return(out)
}

Est <- estimMV(rnorm(100, 1, sqrt(3)))
print(Est$Mean)
print(Est$Var)

Est <- estimMV(runif(100, -2, 2))
print(Est$Mean)
print(Est$Var)
```

4 Basic graphic tools

The usual functions applied to the 2D graphical representations are `plot`, `lines`, `curve` and `points`. Do not hesitate to look at `help(plot)` to get an overview of the numerous opportunities.

4.1 Examples of graphs

Try to change `pch`, `col`, `type`, `lwd` or `lty` arguments. Look also at `xlim`, `ylim`, `main`, `xlab` or `ylab` to decorate the graph.

```
# Discrete representation of f(x) = ln(x^2 + 1/x^2)
X <- seq(-4, 4, by=0.01)
Y <- log(X^2+1/X^2)
plot(X, Y, col="blue")

plot(X, Y, col="blue", pch=3)
plot(X, Y, col="blue", type="l", main="Graph")
plot(X, Y, col="magenta", type="l", lwd=3, lty=2, xlab="Abs. X", ylab="Ord. Y")
```

You may add other graphs on top of the first using `lines` or `curve` with the option `add=TRUE`, having its own properties. Use `points` to add a scatter plot, and use `text` to insert some text in the graph. With `grid`, it is also possible to add a grid in background.

```
# Discrete representations of f(x) = ln(x^2 + 1/x^2) and g(x) = -x^2+6
Z <- -X^2+6
plot(X, Y, col="magenta", type="l", lwd=3, lty=2, xlab="Abs. X", ylab="Ord. Y")
lines(X, Z, type="l", lwd=3, col="red")

# That's... nonsense, really
X <- rnorm(20)
Y <- rexp(20)
plot(X, Y, col="blue", type="p")
points(X+0.1, Y+0.1 , pch=2, col="red")
lines(sort(X), Y, lty=2, col="orange")
text(mean(X), max(Y), "Hello", col="magenta")

# Use of 'curve' to get continuous representations of functions of x
curve(sin(x), from=-2*pi, to=2*pi, col="red", lwd=2, xlim=c(-4, 4), ylim=c(-1, 1))
curve(cos(x), from=-2*pi, to=2*pi, col="blue", lwd=2, add=TRUE)

# Add a grid
grid(col="lightgray", lty="dotted")
```

4.2 Add a legend

It is possible to add a legend to your graph using `legend`, that you manually locate. The properties of each graph appearing in the legend must be specified in the same order, to handle the graphic representation.

```
# Same example as above, with its legend
X <- seq(-4, 4, by=0.01)
Y <- log(X^2+1/X^2)
Z <- -X^2+6
plot(X, Y, col="magenta", type="l", lwd=3, lty=2, xlab="Abs. X", ylab="Ord. Y")
```

```
lines(X, Z, type="l", lwd=3, col="red")

legend("topright", c("f(x)", "g(x)"), col=c("magenta", "red"), lwd=c(3, 3), lty=c(2, 1))
```

4.3 Statistical tools

Histograms, boxplots, regression lines, kernel densities, ... are also easily available using `R`. Here are some examples.

```
# Histogram, density and boxplot of a standard Gaussian sample
X <- rnorm(1000)
hist(X, breaks=15, col="lightblue", border="blue", freq=FALSE, xlim=c(-4,4))
lines(density(X), col="red", lwd=2, lty=2)

boxplot(X, main="Boxplot of X", col=c("gold"))

# Regression line of a scatter plot
X <- 0.5*rnorm(100)
E <- rnorm(100)
Y <- 2 + 2.5*X + E
plot(X, Y, type="p", pch=3)
LinReg <- lm(Y~X)
summary(LinReg)
abline(LinReg, col="magenta", lty=2, lwd=2)
```

5 Try yourself !

We suggest some little activities, to practice.

5.1 Factorial

Create a function `fact(n)` that takes an integer n as argument and returns the factorial $n\,!$ of n. Create another function `recfact(n)` that produces the same result but using a recursive calculation (namely, `recfact(n)` calls `recfact(n-1)`, and so on). Recall that by convention, $0\,! = 1$.

5.2 Average profile

The dataset `nottem` contains the average temperatures at Nottingham (in degrees Fahrenheit) each month between 1920 and 1939. Look at `nottem` and extract the data in a matrix form using for example :

```
> Mat <- matrix(as.numeric(nottem), nrow=20, ncol=12, byrow=TRUE)
```

Build a vector containing the mean values of each month along the year and draw it.

5.3 Piechart

You have a vector `Eff` containing the sizes of some classes (say for example `Eff = c(20, 15, 30, 25)`). Look in the `R` help pages to find a piechart representation of `X` where you can deal with colors, add text, select clockwise or counterclockwise direction, etc.

5.4 Determinant

Create a function `det2d(M)` that takes a square matrix M of size 2 as argument and returns its determinant (without using the `det` function). Create also a function `det3d(M)` that takes a square matrix M of size 3 as argument and returns its determinant by using the `det2d` function.

5.5 Parabola

Create a function `parab(a, b, c)` that takes the coefficients of the parabola $f(x) = ax^2 + bx + c$ as arguments and returns its zeroes (using complex variables if needed). Draw the parabola and add a cross onto the real zeroes.