1 An Introduction to R

Solutions to exercises.

2 Starting and stopping

Download R at

http://www.r-project.org/

install it and start an R session. You get a window with a prompt > where you can type commands. To stop the R session, type q() and hit Return

```
> q()
Save workspace image? [y/n/c]:
```

We do not want to save anything, so type n.

3 Calculator

First of all, R can be used as a simple calculator.

exercise 3.1 Copy the following commands into R. Try some variants and make sure you understand.

```
> 2+4
[1] 6
> 3*5
[1] 15
> 2^3              # same as 2*2*2 or 2**3
[1] 8
> sqrt(9)
[1] 3
> sin(pi)          # R knows about pi
[1] 1.224606e-16   # well, that's really 0!
```

The hash symbol # indicates a comment that is meant to be read by humans. R ignores comments.

The up and down arrows of your keyboard allow you to scroll through previous commands. Once you recalled a command, you can edit it and then execute it.

exercise 3.2 Try out the up and down arrows now; it will save you a lot of typing later! Recall your earlier command `2+4` and change it into `3+4`.

If you want to put multiple commands on one line, you have to separate them with a semi-colon `;`.

```
> 2+4; 3*5
[1] 6
[1] 15
```

If a command is incomplete, `R` will change the prompt from `>` to `+`. If that happens, you can either complete the command, or hit `Esc`.

```
> 3*
+
```

... `R` is now waiting for you to tell it what to multiply 3 with.

Sometimes the answer of a calculation is not a number but rather `TRUE` or `FALSE`,

exercise 3.3 Copy the following commands into `R`. Try some variants and make sure you understand.

```
> 2>4               # is 2 greater than 4?
[1] FALSE

> 3==3              # mind the double ==
[1] TRUE

> 3!=3              # != stands for inequality
[1] FALSE

> (3>2)&(3>4)       # & stands for the logical "AND"
[1] FALSE

> (3>2)|(3>4)       # | stands for the logical "OR"
[1] TRUE
```

4 Variables

The following command makes a new variable called `a` and assigns the value $\sqrt{10} = 3.162$ to it

```
> a  =  sqrt(10)  # same as a <- sqrt(10)
```

By typing the name of a variable and then hitting `Return` we can check its value:

```
> a
[1] 3.162278
```

We can also calculate with variables. The command

```
> b  =  a+5
```

makes a new variable called `b` with value $\sqrt{10} + 5 = 8.162$.

A so-called Boolean variable (after the English mathematician George Boole) takes the value `TRUE` or `FALSE`

```
> c = (3<4)       # c is a Boolean variable
> c
[1] TRUE
```

You can also use Boolean variables in calculations. Then `TRUE=1` and `FALSE=0`. Here is an example of such a calculation.

```
> (3<4) + 3
[1] 4
```

Finally, a variable can also have a character string as value, such as

```
> d = 'hello world'
```

This is very useful, but we won't make much use of it in this course.

5 Vectors

A (numeric) vector is a list of numbers. Vectors are *very* important statistics and in R. There are a few different ways to make them.

exercise 5.1 Try out the following commands one by one. Each time, check the resulting vector x

```
> x = c(1,3,2,6,0)        # c stands for combine or concatenate, I think ...
> x = 1:10
> x = seq(1, 20, by = 3)  # seq stands for sequence
> x = seq(10, 1, by = -2)
> x = rep(1:5,times=3)    # rep stands for repeat
> x = rep(1:5,each=3)     # note the difference with the previous command
```

exercise 5.2 Make the vector $(0, 0.01, 0.02, \ldots, 0.99, 1)$. Also, make the vector $(5, 4, 3, 2, 1)$ Also, make $(1, 2, \ldots, 10, 13, 14, 16)$.

```
x=seq(0,1,0.01)
x=5:1   # or x=seq(5,1) or x=seq(5,1,-1)
x=c(1:10,13,14,16)
```

We can also calculate with vectors. The operations are simply applied to all the elements of the vector.

```
> x = 1:10      # make some vector
> y = 3*x - 12  # do some calculation
> y
 [1] -9 -6 -3  0  3  6  9 12 15 18
> z = (y<0)     # another calculation
> z
 [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

6 Attributes of vectors

exercise 6.1 Try out the following very important R functions and make sure you understand what they do.

```
> x = c(1,3,2,2)
> length(x)
> sum(x)
> prod(x)
> max(x)
> min(x)        # same as -max(-x)
> which.max(x)
> which.min(x)  # same as which.max(-x)
> sort(x)
```

exercise 6.2 The command `x = sample(1:10,10000,replace=TRUE)` produces a vector of 10000 random numbers between 1 and 10. Execute this command, and then count how many fives you got.

```
x = sample(1:10,10000,replace=TRUE)
sum(x==5)
```

exercise 6.3 Now count how often a 1 is followed by a 2.

Here is one way to do it

```
sum(c(0,x)==1 & c(x,0)==2)
```

... but can you understand *why* it works? To see how it works we should use a smaller vector `x`. Run the following 3 lines:

```
x=c(2,3,1,1,3,1,2,4,5)
c(0,x)
c(x,0)
```

7 Subscripting

Subscripting refers to selecting certain elements of a vector. We use straight brackets `[ ]`

```
> y = 3*(1:10) - 12   # make some vector
> y[3]                # get the third element of y
[1] -3
> y[3:5]
[1] -3  0  3
> y[c(1,3,5)]
[1] -9 -3  3
```

Negative selection is also possible

```
> y[-3]                          # leave out the third element
[1] -9 -6  0  3  6  9 12 15 18
> y[-c(1,3,5)]                   # leave out elements 1,3 and 5
[1] -6  0  6  9 12 15 18
```

We can also select with a Boolean vector. We can select all elements of `y` that are positive by

```
> y[y>0]
[1]  3  6  9 12 15 18
```

exercise 7.1 The command `x = sample(1:10,10000,replace=TRUE)` produces a vector of 10000 random numbers between 1 and 10. Select all elements greater than 5

```
x = sample(1:10,10000,replace=TRUE)
x[x==5]
```

exercise 7.2 Use the command `which` to do the same thing you did in the previous excercise. If you don't know how `which` works, check the help file.

```
x = sample(1:10,10000,replace=TRUE)
ind=which(x==5)               # take a look at the vector "ind"
x[ind]
```

8 Matrices and data frames

A matrix is a rectangular array of numbers. We can use the `matrix` command to construct a matrix out of a vector.

exercise 8.1 Compare the following commands.

```
x = c(1,8,6,3,4,9,10,5,7,2)
A = matrix(x,nrow=2)
B = matrix(x,nrow=2,byrow=TRUE)
C = matrix(x,nrow=5)
D = matrix(x,nrow=5,byrow=TRUE)
```

Do you understand the differences between the matrices A, B, C and D?

Subscripting in matrices works very similar to vectors. The only diference is that the elements of a matrix a matrix are indexed by two numbers.

exercise 8.2 Try out the following commands to see how it works.

```
A[1,1]
A[1,2]
A[2,1]
A[2,3]
A[,4]  # the whole fourth column
A[2,]  # the whole second row
```

A matrix is like a data set, but it is limited in that it can store only numbers. A `data.frame` a like a matrix, but it can also store strings ("words") and dates.

exercise 8.3 To make a data frame, execute the following commands

```
id = 1:5
treatment = c('case','case','case','control','control')
outcome = c(6.2, 10.2, 13.2,  7.8, 10.8)
data=data.frame(id=id,treatment=treatment,outcome=outcome)
```

Check out what data looks like.

Subscripting for data frames works the same as for matrices. You can also select a column from a data frame by using the $-sign.

exercise 8.4 Verify that the following commands yield the same result.

```
data$outcome
data[,3]
```

Now extract the outcome for all controls from the data frame.

```
data$outcome[data$treatment=='control']
data[data$treatment=='control',3]
```

9 Getting help

We have seen a few `R` functions, but of course there are many more. If you know the name of a function, you can ask `R` for help on how to use it by typing

```
> help(seq)    # same as ?seq
```

You will get much more information than you need, but usually at the end of the help file, there are a few nice examples.

Google is perhaps even more useful. Type something like: "R sequence of numbers" and you will probably find what you need, even if you did not know about `seq`.

10 Input and output: `load` and `cat`

The `R` command `cat` (short for concatenate) writes to the screen.

exercise 10.1 Try out the following code

```
> x=22/7
> cat("The value of pi is approximately",x,"but better approximations do exist.\n")
```

The `"\n"` forces a new line. Leave it off, and see what happens.

The `cat` command can also be used to wite to a file, but other commands are typically more suitable. There are also commands that read files into `R` which is very useful for getting data. One of these commands is `load` which allows you te read text files.

11 Plotting: `plot` and `points`

`R` has a sophisticated graphical system, which has been extended through the package `ggplot2`. Now, you can make almost any graph you can think of and make it look professional. Here, we will only mention the two most basic plotting commands that are available in base `R`: `plot` and `points`.

exercise 11.1 Try out the following code

```
> x = seq(0,2*pi,by=0.1)
> plot(x,sin(x))
```

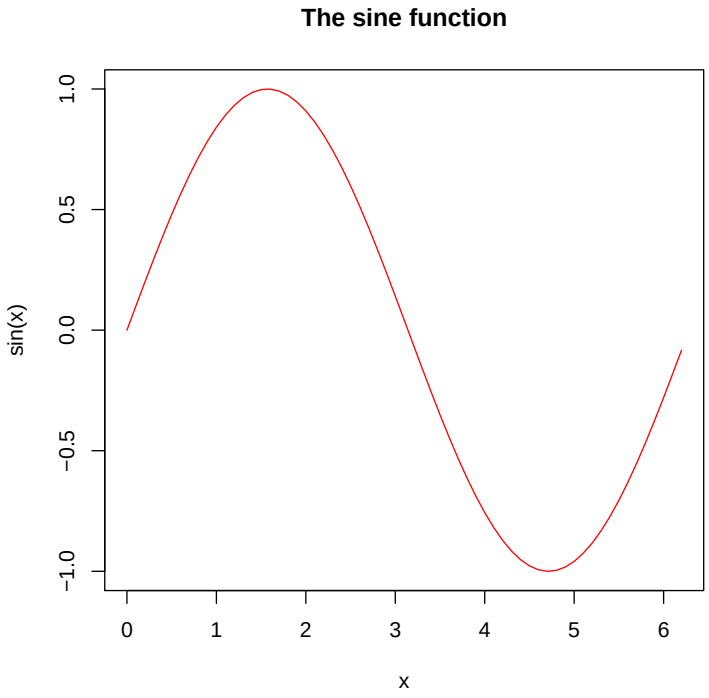
Now change the last command to

```
> plot(x,sin(x),type='l')
```

and observe the difference. Note: `'l'` is an ℓ and not a one.

exercise 11.2 The `plot` function has many other options besides `type`. Read the help file, and reproduce exactly the plot below. Many other colors besides `red` are available, such as `black`, `blue`, `green`, `purple`, et cetera. To save your plot, activate the plotting window and go the `file` drop-down menu of `R`. Try this out.

```
x = seq(0,2*pi,by=0.1)
plot(x,sin(x),type='l',col='red',main='The sine function',ylab='sin(x)')
```



The command `plot` always starts a new graph. Sometimes, we want to add more elements to an existing graph. To do that, we can use the command `points`.

exercise 11.3 Try out the following code

```
> x = seq(0,2*pi,by=0.1)
> plot(x,sin(x),type='l')
> points(x,cos(x),type='l',col='red')
```

exercise 11.4 Plot the function $f(x) = x^2$ on the interval $[-5, 5]$. Next, add the function $g(x) = |5x|$ to your graph. Use the `R` command `abs` to compute the absolute value.

exercise 11.5 Think of some function yourself and plot it. Think of another function, and add it to your plot.

12 Scripting

A *script* is a sequence of commands. Of course you can always type all your commands one by one at the `R` prompt, but that is not very practical. You should write your scripts using an editor. To start a new script, go to the drop-down `file` menu of `R` and select `new script`. Type your list of commands and hit `Ctrl-s` to save it. You will be prompted to give your script a name. Choose any name you like, but end it with `.r` or `.R`.

Now, you can select some or all the lines you wrote and hit `Ctrl-r`. The commands you selected are sent to the `R` processor and executed.

exercise 12.1 Here is a little script

```
x=1
y=2
cat(x,"+",y,"=",x+y,"\n")
```

Read the instructions above and save these three lines in a file. Select some or all lines and send them to `R`. Now close your script, and then open it again by going to the drop-down `file` menu of `R` and selecting `open script`.

From now on, you should always work with and editor and use `Ctrl-r` to run some commands. And do yourself a favour; use the hash symbol `#` to add lots of comments. That way you'll know what you were thinking when you look at your code later!

There are a few specialized `R` editors that you might like, such as the freely available RStudio and Tinn-R.

exercise 12.2 Download RStudio or Tinn-R (or both), install them and open your script. Figure out how you can select a few lines and execute them. A little time invested now will make your life much easier later!

13 Compiling reports

exercise 13.1 Compile the report and appreciate the beauty of it!

14 `for` loop

Unlike humans, computers do not mind doing essentially the same thing many times. The `for` loop is one way to implement such behaviour.

exercise 14.1 Here is a simple `for` loop. Try it out.

```
for (i in 1:10) {
    cat("hello",i,"\n")
}
```

Note how the "body" of the `for` loop is enclosed in curly brackets. I also indented the body of the loop. You don't have to do that, but when you have many loops or even loops within loops, indentation is a good way to keep your script readable.

exercise 14.2 Now try out another `for` loop

```
for (i in c(1,4,2,5,1,1)) {
    cat("hello",i,"\n")
}
```

Do you understand what is happening?

exercise 14.3 Now use a `for` loop to write the squares of the numbers 3 through 12 to the screen.

Of course, we did not really need to use a `for` loop to get those squares; `(3:12)^2` does the same thing, and faster. Here is a more interesting challenge for you:

exercise 14.4 In the Fibonacci sequence, every number is equal to the sum of the two preceding numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, et cetera. We can use `R` to compute the first 20 Fibonacci numbers. Start by making a vector of length 20.

```
fib=rep(0,20)
fib[1]=0; fib[2]=1   # set the first two Fibonacci numbers
```

Now change elements 3 through 20 of `fib` with a `for` loop. Make a plot of the Fibonacci sequence.

```
fib=rep(0,20)
fib[1]=0; fib[2]=1   # set the first two Fibonacci numbers
for (i in 3:20){
   fib[i]=fib[i-2]+fib[i-1]
}
cat(fib,'\n'); plot(fib)
```

15 `if` statement

Sometimes we want commands to be executed *only* under certain conditions. We can use the `if` statement to accomplish this. Here is a simple example which you should try out.

```
for (i in 1:10) {
    cat(i,i^2,"\n")
    if (i == 9) {
        cat("just one more:\n")
    }                          # this bracket closes the if statement
}                              # this bracket closes the for loop
```

exercise 15.1 In an earlier exercise, you were asked to make a vector of 10000 random numbers between 1 and 10 and to count the number of fives. That is easy:

```
x = sample(1:10,10000,replace=TRUE)
count=sum(x==5)
```

Here is another way to do it: First set `count=0`. Then use a `for` loop to go through all the elements of `x` and whenever we find a five, we add one to our count. Implement this approach.

```
count=0
for (i in 1:10000){
  if (x[i]==5) {
     count=count+1
  }
}
```

Here is a third way to do it. Instead of using `if` statement, add a Boolean variable to the counter.

```
count=0
for (i in 1:10000){
     count=count+(x[i]==5)
}
```

exercise 15.2 Again, use the vector of 10000 random numbers. Use a `for` loop to count how often a 1 is followed by a 2.

```
count=0
for (i in 1:9999){
     count=count+(x[i]==1 & x[i+1]==2)
}
```

or, almost the same

```
count=0
for (i in 2:10000){
     count=count+(x[i-1]==1 & x[i]==2)
}
```

Of course, this only works if the vector `x` has 10000 elements. a more genral approach is as follows

```
count=0
n=length(x)
for (i in 2:n){
     count=count+(x[i-1]==1 & x[i]==2)
}
```

16 Functions

We have seen a few `R` functions, but of course there are many more. You can also make your *own* functions. Here is an example

```
add3 = function(x) {
   return(x+3)
}
```

You must run this little script once, and then you can use your new function. For instance

```
> add3(2)
[1] 5
```

Here is another example with two arguments (inputs)

```
myfun = function(x,y) {
   return(3*x+2*y+5)
}
```

exercise 16.1 Write an R function that computes the square of a number.

```
square = function(x) {
   return(x^2)
}
```

exercise 16.2 Write an R function that counts the number of fives in a vector.

```
count.fives = function(x) {
   return(sum(x==5))
}
x = sample(1:10,10000,replace=TRUE)
count.fives(x)
```

17 Other commands

We have now discussed almost all the R commands we will use in this course, but there are just a few more:

- We are going to use two mathematical functions: the logarithm and the exponential function. These are called `log` and `exp` in R. Also, we will use R to find the maximum of a function. We have already seen the command `which.max` but we will also use the command `optim`.

- For probability theory we will use `R` to compute probabilities and to sample from probability distributions. For instance, to sample from the binomial ditribution we will use `rbinom`.
- For statistics we will use `R` to make nice plots such as histograms `hist` and boxplots `boxplot`. We will also perform statistical tests. In particular, we will discuss `prop.test` and `t.test`. Often statisticians are asked how large a sample should be to be able to draw valid conclusions. We will `power.prop.test` and `power.t.test` to answer such questions.

Don't worry about these additional commands now; we'll discuss them later.