An Introduction to R

The following pages will give you a simple introduction to the software package R. As well as attempting the various tasks, you should type in each command that is shown next to the symbol

```
>
```

(which we refer to as the **command prompt**), and press return. You should check that the output on the screen matches the output shown here.

1 Starting R on the University network

You will need to start off by starting R on your computer. If R does not appear on the menu, you may need to install it; your tutor should be able to help.
You can also download R for your own computer. Various versions are available from `https://www.r-project.org/`.

2 Using R as a calculator

To do addition, subtraction, multiplication and division in R use the symbols `+`, `-`, `*` and `/` respectively. For example, if you type `5 * 6` at the command prompt, and press return, you will see

```
> 5 * 6

[1] 30
```

The symbol `^` is used for raising a number to a power. For example, to calculate 2^4, try the following

```
> 2 ^ 4

[1] 16
```

You can use many standard mathematical functions such as `sin()`, `cos()`, `tan()`, `log()` and `exp()` (for e^x).

- You must use the brackets: `cos(2)` will work, `cos 2` will not.
- The R command `log()` calculates the natural logarithm $\ln x$, not $\log_{10} x$. Use `log10()` for log base 10.
- Angles are specfied in radians, not degrees.
- R will interpret the word `pi` as the constant π.

Some examples:

```
> cos(pi)

[1] -1

> exp(2)

[1] 7.389056

> log(2.718282)

[1] 1

> log10(100)

[1] 2
```

R will follow the BODMAS[1] rule for ordering operations. For example:

```
> 4 / 2+2

[1] 4

> 4 / (2+2)

[1] 1

> 16 ^ 1 / 2

[1] 8

> 16 ^ (1 / 2)

[1] 4
```

Task 1. Use R to calculate $8^{\frac{1}{3}}$, $\sin(\pi/2)$, $\ln 4$ and e^{50}. If you have entered the commands correctly, you should get, respectively, the outputs

```
[1] 2
[1] 1
[1] 1.386294
[1] 5.184706e+21
```

- R will only display a limited number of digits, so $\ln 4$ only equals 1.386294 correct to seven significant figures: try calculating $(\ln 4) - 1.386294$ in R.
- The R output `5.184706e+21` should be read as "5.184706 multiplied by 10 to the power of 21".

[1] Brackets, Orders (powers/roots), Division, Multiplication, Addition, Subtraction

3 Defining and using variables

You can assign a numerical value to what we refer to as a *variable*, and then use the variable within various R commands. For example

```
> x <- 3
```

defines a variable called x, which takes the value 3. You won't see any output when you type this command, but if you type the variable name on its own, R will tell you its value

```
> x

[1] 3
```

You can now use the variable in a command:

```
> 2 * x

[1] 6
```

Task 2. Assign the number 2.718282 to the variable e, then evaluate ln e. (If you are not sure how to evaluate ln e in R, read through Section 2 again).

(Note: in some other programming languages you use = to assign a value to a variable, e.g. x = 3. In fact this will work in R as well, but R programmers tend to prefer x <- 3 as above.)

4 Vectors

You can define a vector variable using the command c( ), with a list of the elements in your vector, separated by commas, inside the brackets. For example, to create a vector of the numbers 2, 4, 6, 8, 10, and assign it to a variable x type

```
> x <- c(2, 4, 6, 8, 10)
```

If you display the vector, by typing its name, you will see

```
> x

[1]  2  4  6  8 10
```

You can also use vectors within commands. Some commands operate on each element of the vector. For example:

```
> 2 * x

[1]  4  8 12 16 20
```

You can select individual elements of vectors, by specifying the element number inside square brackets. For example, to get the fourth element of x, type

```
> x[4]

[1] 8
```

You can assign values to individual elements of vectors. To change the fourth element of `x` to 9, type

```
> x[4] <- 9
```

and to check that this has worked, type

```
> x

[1]  2  4  6  9 10
```

If you define a second vector of the same size, you can do pair-wise operations between the elements of the two vectors. For example:

```
> y <- c(5, 4, 3, 2, 1)
> x * y

[1] 10 16 18 18 10

> x ^ y

[1]  32 256 216  81  10
```

Various commands in R are designed specifically for vectors. For example, to add up all the elements of a vector, use the `sum()` command:

```
> sum(x)

[1] 31
```

You can test whether each element of a vector satisfies a particular condition. For example, to test whether each element of `y` is less than 4, we can do

```
> y < 4

[1] FALSE FALSE  TRUE  TRUE  TRUE
```

If we `sum()` the result of this, R will interpret each `TRUE` as a 1 and each `FALSE` as a 0; we count how many elements of `y` are less than 4.

```
> z <- y < 4
> z

[1] FALSE FALSE  TRUE  TRUE  TRUE

> sum(z)

[1] 3
```

Task 3. Defined `x` to be a vector with elements 3, 1, 7, 3 (in that order). Guess what each of the following eight commands will do in R: complete as many rows of the following table as you can, *before typing in any further commands.*

R command	result
`min(x)`	
`max(x)`	
`prod(x)`	
`sort(x)`	
`sort(x, decreasing = TRUE)`	
`length(x)`	
`unique(x)`	
`sum(x < 6)`	

Now try each command, to see if your guesses were correct. Without re-defining the whole vector, change the 2nd element of `x` to 8, and re-do the last command.

You can create sequences of integers by typing the first and last integer, separated by a colon. For example:

```
> 3:12

 [1]  3  4  5  6  7  8  9 10 11 12
```

Task 4. By defining appropriate vectors, use R to calculate

$$\sum_{x=1}^{100} x \text{ and } \sum_{y=0}^{100} \frac{1}{2^y}.$$

Your answers should agree with the results from the following two commands (why?)

```
> 100 * (100 + 1) / 2

[1] 5050

> (1 - 0.5 ^ 101) / (1 - 0.5)

[1] 2
```

5 A simulation experiment

R can be used to simulate random processes, and here we consider a very simple example: rolling a six-sided die. First define a vector of the six possible outcomes:

```
> sides <- 1:6
```

Then, to simulate one die roll, use the `sample` command.

```
> sample(sides, 1)
```

This will choose one element of `sides` at random, with each element of `sides` having the same chance of being selected.

> You can use the up and down arrow keys on the keyboard to move between previous commands. Use the up arrow key to run the previous command again, and repeat a few times.

To simulate 10 dice rolls, type

```
> sample(sides, 10, replace = TRUE)
```

The argument `replace = TRUE` tells R that an element of `sides` can be selected more than once. (NB you can use `= T` rather than `= TRUE`.) If you want to count how many times a particular number, 3 say, appears, first store the result of the `sample` command:

```
> x <- sample(sides, 10, replace = TRUE)
```

Typing the following will tell you which elements of `x` are equal to 3:

```
> x == 3
```

and to count how many elements of `x` are equal to 3, type

```
> sum(x == 3)
```

To get the proportion of times a 3 was ‘rolled’, type

```
> mean(x == 3)
```

(Note that here, the command `mean(x == 3)` is equivalent to `sum(x == 3) / 10`). To tabulate how many times each number was rolled, type

```
> table(x)
```

and to plot the results in a bar chart, type

```
> barplot(table(x))
```

5.1 Simulating rolling two dice

We can simulate rolling two dice by using the `sample` command again, and then assigning the result to another variable. Try the following

```
> y <- sample(sides, 10, replace = TRUE)
```

and then

```
> cbind(x, y)
```

to see each pair of simulated dice rolls. To count, for example, the number of times we have rolled two sixes, try

```
> sum(x + y == 12)
```

and check the result with your result from the command `cbind(x, y)`

Task 5. Suppose a six sided die is to be rolled 1000 times. Guess the proportion of times each 'event' in the table below would happen, and fill in the second column.

event	proportion of times occurred (guessed)	proportion of times occurred (observed)
die roll is a 6		
die roll is greater than 2		

Now use R to simulate 1000 dice rolls, and fill in the third column of the table. Compare your guesses with what you observed.

Now suppose two six-sided dice are to be rolled together 1000 times. Guess the proportion of times the two dice rolls would sum to 11.

event	proportion of times occurred (guessed)	proportion of times occurred (observed)
two dice sum to 11		

Simulate 1000 rolls of two dice, and compare the proportion you observe with your guess.

We'll now switch from using R to RStudio. Shut down R with the command

```
> q()
```

(select Don't save if asked to save the workspace image).

6 RStudio

RStudio is a 'front-end' for R that is nicer to use. It can be downloaded for free from `https://www.rstudio.com/` (but you will need to download R first), and it is also on the university network. To start RStudio, go to Start (Windows 10 icon) » RStudio » RStudio

7 Script windows

If you intend to use a series of commands during a session, you should use a script window, as you will find it easier to correct any mistakes or make changes to your commands, and you can save your work afterwards.
To open a script window in RStudio, select File » New File » R Script from the menu bar.

7.1 Using a script window

When you type a command in the script window, nothing will happen. To get R to run your commands, highlight them with the mouse, and then press Ctrl and Enter together, or click on the Run button at the top of the script window. Your commands and any output will appear in the R console window, as if you had typed them at the command prompt. You can highlight and run groups of commands, as well as sections within commands (if it makes sense to do so). Anything you highlight and run (using the Run button or Ctrl and Enter) will be copied and pasted into the R console window.

Task 6. Open a new script window, and type in the commands (one on each line)

```
x <- 1:10
sum(x)
sum(x^2)
```

Highlight all three lines with the mouse, and press Ctrl and R together. Check the output in the R console window. Highlight `x` on its own, and press Ctrl and R, and inspect the result in the R console window. Now change the first line to `x <- 11:20` and run all three lines again.

7.2 Saving and loading scripts

To save a script, select the script window and go to File > Save as... Choose a filename that ends with `.R` as it will make the script easier to find when you want to load it. Scripts can be loaded by selecting File > Open File... on the menu bar.

Task 7. Create a folder called MAS113 in your U drive. Save your script from Task 6 in this folder, using the name `worksheet1.R`
Look in your MAS113 folder to see that the file is there, then shut down RStudio. Re-start RStudio, and load your script.

From now on, always use RStudio, and use a script window rather than the command line.
Do not type any commands at the command prompt!

(I will still refer to 'R' rather than 'RStudio' in these notes, because RStudio is simply an interface for using the language R.)

8 Set operations

We can define (finite) sets in R, and do the usual set operations. Try the following example. First define the sets

$$\begin{aligned} A &= \{\text{red, green, blue}\}, \\ B &= \{\text{green, blue, yellow}\}, \\ C &= \{\text{blue, green, red}\}. \end{aligned}$$

using the commands

```
A <- c("red", "green", "blue")
B <- c("green", "blue", "yellow")
C <- c("blue", "green", "red")
```

Make sure you use capital letters for the set names (all inputs to R are case sensitive). The quote marks tell R to interpret the colours as 'string' variables (words, rather than variables representing numerical quantities).

Task 8. First complete the following table, without running any commands in R. Then run the commands to check your answers are correct.

Set operation	R command	result
$A \cup B$	`union(A,B)`	
$A \cap B$	`intersect(A,B)`	
$A \setminus B$	`setdiff(A,B)`	

Note that sets A and C are the same, as the order we list the elements does not matter. Try the following

```
setequal(A, B)
setequal(A, C)
```

9 R packages

There are a huge number of 'packages' that provide additional R commands. (Researchers who develop new statistical methods often write R package to make their methods available to everyone.) A small number are pre-installed but we will now install one that isn't: `gtools`.

1. From the menu bar in RStudio, go to Tools ⟩ Install package(s)...
2. In the middle box, type `gtools` and click Install.
3. In the commands window (or script window), run the command

   ```
   library(gtools)
   ```

The next time you use R, if you want to use the `gtools` package again, you will only need to repeat step 3.

10 Permutations and combinations

In this section, we'll consider a simplified version of the National Lottery (simplified so it's easier for you to visualise the results).

- A lottery machine has six balls, numbered 1 to 6.
- On your ticket, you choose two different numbers from 1 to 6.
- Two balls are drawn from the lottery machine, and if the numbers match those on your ticket, you win.

Suppose you choose the numbers 1 and 3. Assuming each possible draw is equally likely, what is the probability you win? We'll use R to calculate the answer.
Firstly, how many possible draws are there? We can use R to list all the possibilities. Try the following command (you will need the R library `gtools` installed).

```
permutations(6, 2)
```

You'll see that this produces a list of 30 possible draws. If you have studied permutations before, you'll know that the number of ways of drawing x items out of n, where the order matters is given by

$$^nP_x = \frac{n!}{(n-x)!}$$

In this case, we have $^6P_2 = 6 \times 5 = 30$. Looking at the list produced by R, we see that there are 6 choices for the first number, and for each of these choices, there are 5 choices for the second number, so there are 6×5 choices in total.
In how many of these draws do you win? You can get R to count this for you. Try

```
draws <- permutations(6, 2)
ticket <- c(1, 3)
apply(draws, 1, setequal, ticket)
```

The last command works through each row of the array `draws`, and uses the `setequal` command to see whether the row contains the same elements as `ticket`. The output is a list of 30 results, either `TRUE` or `FALSE`. Elements 2 and 11 of this list are `TRUE`, as rows 2 and 11 of `draws` have the same elements as `ticket`. To count how many `TRUE` results there are, try the following command

```
sum(apply(draws, 1, setequal, ticket))

[1] 2
```

This gives an output of 2, so the probability of winning is 2 out of 30.
Alternatively, we can note that it doesn't matter in what order the numbers are drawn; we would have won whether the draw was 1 then 3, or 3 then 1. How many possible draws are there, if the order doesn't matter? We can again use R to list the possibilities. Try the following command

```
combinations(6, 2)
```

This produces a list of 15 combinations. If you have studied combinations before, you'll know that the number of ways of drawing x items out of n, where the order does not matters is given by

$$^nC_x = \frac{n!}{x!(n-x)!}.$$

We can evaluate this function in R with the command `choose()`. For example, to calculate $^6C_2$:

```
choose(6, 2)

[1] 15
```

We can derive this expression from ${}^nPx$. For each combination, there are $x!$ ways of arranging the items into order, so for every combination, there are $x!$ permutations:

$${}^nP_x = x!{}^nC_x.$$

So, for each of the 15 combinations, the numbers could be drawn in two different orders, giving 15×2 permutations. Returning to the chances of winning, assuming each combination is equally likely, there is only one combination matching the ticket, so the probability of winning is 1 out of 15 (the same result as before).

Task 9. Suppose the lottery machine has ten balls, numbered 1 to 10. On your ticket, you choose three different numbers from 1 to 10. Three balls are drawn from the lottery machine, and if the numbers match those on your ticket, you win. Produce two lists of all the possible draws, one where the order matters and one where it doesn't, and calculate the probability that you win if your ticket is 4,5,6 (and the order doesn't matter).

11 Plotting graphs

You can do various plots in R. Here's an example of plotting a cosine curve:

```
x <- 1:10
plot(x, cos(x))
```

If you try these two commands, you'll see that R has only plotted 10 points. To join them up, you need to add the *argument* `type="l"` to the plot command (`"l"` for **l**ine):

```
plot(x, cos(x), type = "l")
```

> Make sure you type the letter l for **l**ine: `type = "l"`, not the number one: `type = "1"`, otherwise you'll get an error message:
>
> ```
> Error in plot.xy(xy, type, ...):  invalid plot type '1'
> ```

You'll see that the 10 points have been connected by straight lines, which doesn't give a smooth looking curve. To get a smoother looking curve, first define more points on the x-axis:

```
x <- seq(from = 1, to = 10, length = 100)
```

This creates a sequence of 100 points, evenly spaced, between 1 and 10. (Type `x` to see the sequence). Now re-do the plot:

```
plot(x, cos(x), type = "l")
```

Task 10. For x ranging from -4 to 4, plot (as a smooth looking curve) the function

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right).$$

You can add more arguments to have more control over how the plot appears. In a script window, run the commands

```
x <- seq(from = -pi, to = pi, length = 100)
plot(x, cos(x), type = "l", lwd = 1, xlab = "label 1",
     ylab ="label 2", main ="title")
```

(In a script window, you can break a long command over multiple lines: when typing, press return after any comma, e.g. after `"label 1",`)

The argument `lwd` refers to the thickness of the line, and increasing the value from 1 will give a thicker line. Try `lwd = 2`. You can specify the axes labels and title by changing the text `label 1`, `label 2` and `title` to anything you want. If you don't put anything inside the quote marks, the label will be left blank.

To plot another function on the same axes, use the `lines` command:

```
lines(x, sin(x), col = "red", lty = 2, lwd = 2)
```

(re-run both the plot command and the new `lines` command). The option `lty = 1` will give a solid line. Experiment with the values 3,4,5 for the `lty` option. You can add a legend to the plot with the command

```
legend("topleft", c("cos x", "sin x"),
       col = c("black", "red"), lty = c(1, 2))
```

where the four arguments are the position of the legend, the legend labels, the colours of the lines, and the line styles used. If you have changed the thicknesses of the lines, you will need to include an argument such as `lwd = c(2, 2)` (if you chose 2 to be the thickness of each line).

You can add shaded regions (between the curve and the line $y = 0$) with the `lines` command, by including the argument `type = "h"`. For example:

```
x <- seq(from = -1, to = 0, length = 100)
lines(x, cos(x), type = "h")
```

Finally, you can make the fonts larger, by including in your script the command

```
par(ps = 15)
```

before the first plot command. You can choose alternative numbers to 15, to get smaller or larger fonts.

Task 11. On a single set of axes, for x ranging from 0 to 3, plot the two functions $2e^{-2x}$, and $1 - e^{-2x}$, with a larger font size and line widths than the default setting.

- Use a blank label for the y-axis.
- Use different colours and line styles for each function.
- Add a legend, using the names "pdf" and "cdf" respectively (if you have studied probability before, you may recognise these two functions as the **p**robability **d**ensity **f**unction and **c**umulative **d**istribution **f**unction of the exponential distribution with rate parameter 2).
- Draw a shaded region under the function $2e^{-2x}$ between $x = 1$ and $x = 2$. What is the area of this shaded region?

12 R Markdown

R Markdown is a powerful system for producing reports that include analyses using R (and other languages). It can produce reports in the format of a Word document, pdf file, or web page. (You don't need to know anything about producing web pages using, say, html: R Markdown will do all the hard work for you).

> If you do any project work involving R, it will almost always be quicker and easier to use R Markdown than it will be to copy and paste/import R plots/code/results into other software packages.

To use R Markdown, you will need to install a package (once only), using the command

```
install.packages("rmarkdown")
```

To create a new R Markdown document, you can select File › New File › R Markdown... or File › New File › R Notebook (“R Notebooks” are essentially R Markdown documents.)

Task 12. Download the file `MAS113-Rmarkdown-example.rmd` from the course web page, and open it in RStudio. By following the instructions in the file, produce a web page, a pdf document, and a Word document, that each present your solution to the previous task (both the R commands, and the resulting plot.)