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 in this workbook.

1 Starting R on the University network

Click on Start > All programs > R 3.1.1 > R i386 3.1.1
(The actual numbers may be higher if R has been updated).

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, type `2^4`:

```
> 2^4
[1] 16
```

You can use many standard mathematical functions such as `sin`, `cos`, `tan`, `log` and `exp` (for e^x). Note that angles are specified in radians, and the default base for the `log` command is e. (Use `log10` for log base 10). R will interpret the word `pi` as the constant π. Some examples :

```
> cos(pi)
[1] -1
> exp(2)
[1] 7.389056
> log10(100)
[1] 2
```

Note how the brackets are used in each function. Typing `cos pi`, `log10 100` etc will not work.

Task 1. Use R to calculate $8^{\frac{1}{3}}$, $\sin(\pi/2)$, $\ln 4$ and e^{50}. (Note that the R output `e+` should be read as "multiplied by 10 to the power of").

3 Using the arrow keys

The up and down arrow keys on the keyboard can be used to move between commands previously entered at the command prompt. Try using them now.

4 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).

5 Defining and viewing a variable in one step

You can define a variable and view the result in one step, by putting brackets around the command. Try the following.

```
> (a<-1000^0.5)
> a+10
```

6 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
```

Task 3. Defined `x` to be a vector with elements 3, 1, 9, 7, 3 (in that order). Guess what each of the following eight commands will do in R, then try them: `min(x)`, `max(x)`, `prod(x)`, `sort(x)`, `unique(x)`, `length(x)`, `x<6` and `sum(x<6)` . Without re-defining the whole vector, change the 2nd element of `x` to 8, and re-do the last two commands.

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. Use R to calculate

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

(The first sum is an arithmetic series, and the second sum is a geometric series. Do you know the formulae for evaluating arithmetic and geometric series? If you do, check that these give the same results as you get by using R to do the calculations directly.)

7 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<-c(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. Try running this command a few times (type the command once, then use the up arrow on the keyboard to bring up the command again). To simulate 10 dice rolls, type

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

The argument `replace=T` tells R that an element of `sides` can be selected more than once. 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=T)
```

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 tabulate how many times each number was rolled, type

```
> table(x)
```

and to plot the results in a bar chart, type

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

Task 5. Suppose a six sided die is rolled 600 times. How many times would you expect the outcome to be a 6? Simulate 600 dice rolls, and count how many sixes you get. Find the proportion of sixes by dividing the total number of sixes by 600. Repeat your simulation a few times. What do you think will happen to the proportion of sixes if you simulate 6,000,000 rolls rather than 600? Try it!

8 RStudio

RStudio is a 'front-end' for R that is nicer to use. It can be downloaded for free from `www.rstudio.com` (but you will need to download R first), and it is also on the university network. If you are currently using R, shut it down, then go to

Start > All programs > RStudio > RStudio

9 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** > **Rscript** from the menu bar.

9.1 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 R 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 Ctrl+R) 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)

```
8^1/3
8^(1/3)
x<-c(1:10)
sum(x)
sum(x^2)
```

Highlight all five lines with the mouse, and press Ctrl and R together. Check the output in the R console window. (Why do the first two commands give different results?) Highlight `x` on its own, and press Ctrl and R, and inspect the result in the R console window. Now change the third line to `x<-c(11:20)` and run the last three lines again.

9.2 Saving and loading scripts

To save a script, select the script window and go to **File** > **Save as...** on the menu bar. 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 1 in this folder, using the name `worksheet2.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.)

10 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. Try the following set operations in R

Set operation	R command
$A \cup B$	`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)
```

11 R libraries

There are a huge number of 'libraries' that provide additional R commands. (Researchers who develop new statistical methods often write R libraries 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` library again, you will only need to repeat step 3.

12 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). Suppose the 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))
```

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 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).

12.1 Permutations and combinations for non-numerical items

We can get R to list permutations and combinations for non-numerical items, using a few extra commands. Suppose we define a list of five England bowlers:

```
bowlers<-c("Anderson","Broad","Finn","Jordan","Woakes")
```

What are the different combinations of three bowlers out of five? (And how many are there?) In a script file, under the definition of `bowlers`, add the lines

```
x<-combinations(5,3)
bowler.list<-bowlers[x]
(bowler.combinations<- matrix(bowler.list,ncol=3))
```

If you inspect the value of `x`, you will see the list of combinations in numerical form. Think of this as
bowler 1, bowler 2, bowler 3
bowler 1, bowler 2, bowler 4
etc.
Now imagine taking the columns of `x` and arranging them in a long list:

1,1,1,1,1,1,2,2,2,3, 2,2,2,3,3,4,3,3,4,4, 3,4,5,4,5,5,4,5,5,5.

If you inspect `bowler.list`, you'll see a list of the corresponding bowler names: `"Anderson"` 6 times, then `"Broad"` 3 times and so on. The last command takes this list of bowler names, and arranges it into a matrix with 3 columns (`ncol=3`), filling up one column at a time.

Task 10. Given the number of combinations, how many permutations of three bowlers out of five are there? Modify your commands above to produce a list of all the possible permutations of three bowlers out of five.

13 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, add the *argument* `type="l"` ("l"for line, not the number 1) to the plot command:

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

(If you get the error message
`Error in plot.xy(xy, type, ...)  :  invalid plot type '1'`
it's because you didn't do "l" for line in `type="l"`) 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 11. 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")
```

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))
```

You can change the appearance of the line by specifying a colour and line style (and width):

```
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 12. 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?

Copy your graph into a Word document, and save the Word document in your MAS113 folder. To do this in RStudio, click on **Export** in the plot window, then choose **Copy to clipboard**. Select the **Copy as Metafile** option before you actually copy the graph to the clipboard. Then paste the graph into Word.