MAS115 R programming 2016-17
Exercises 1

1 First steps

1.1 Creating a workspace and a script file

It is very important that all the programming that you do is filed neatly on your computer so that you can access it easily at a later date. You should create separate folders for each significantly different bit of work and store all the relevant scripts, datasets and plots within that folder. All of these files should have memorable names indicating what they contain. If you do not do this then you will soon end up unable to remember the purpose of large numbers of your files.

1.2 Starting your session

1. Before loading R please do the following:

   - Use Windows Explorer to create a new folder in your `U://My Documents` drive entitled `MAS115`.
   - Within this new folder create a sub-folder called `RExerciseLabs`
   - Within this new folder create a sub-folder called `Lab1`

   We will use the `RExerciseLabs` folder to do all the R work for these practicals and create separate subfolders for each specific class. It will store all the code we write in script files (and any output we create).

2. Start **RStudio** from the application menu.

3. We want to work in the folder that we just created. To do this we need to change what is known as the working directory.

   Within **RStudio** select **Session** > **Set working directory** > **Choose directory ...** on the menu bar. Browse to the folder you just created and select it.

   N.B. To check which directory we are currently working in we can use the command `getwd()`.

4. Create a script window by selecting **File** > **New file** > **R script** from the menu bar. We will write *all* of our commands in this editor and not directly into the command window. To run any command or selection of commands, highlight the desired commands using the mouse and either press `Ctrl+R` or click on the `Run` button at the top of the script window. We will want to save this script file as we edit it and probably reload it later.

   - To save a script, select the script window and go to **File** > **Save as...** on the menu bar. Call the script `Exercises1.R`. When entering any script filename, it is best to add the .R extension at the end, as it will make the script easier to locate.

- If you wish to load a script at a later date then this can be done by selecting **File** > **Open File...** on the menu bar.

We are now ready to proceed with the rest of the class. Again all code should be written in the script window `Exercises1.R` and nothing should be typed directly to the command line.

1.3 Commenting

Whenever you program, your code must be commented so that someone—often a future 'you'—can look at it and figure out what the various parts do. When you do homework and the assignments we will check that you have added sufficient commenting to your code for another user to understand. This will be part of the assessment criteria and you will be marked down if no such comments exist.

In order to create a comment in R, you use the `#` character. Anything occurring later on the same line of code will not be read by the command line. For example try running

```
x <- 1:3
x <- x + 1  Add 1 to every element in x
```

and you will find that it doesn't work—R gives you an error shown by some red text. However if instead you add a `#` command when you want to add a comment then it will.

```
x <- x + 1 # Add 1 to every element in x
```

As an example of some commented code (try this out) you might end up with something like

```
x <- x + 1 # Add 1 to every element in x


# Find the 'empirical' (or 'population' or 'uncorrected') variance of a vector
EmpVar <- function(x)
{
 n <- length(x) # Find the length of x
 (sum(x^2) - n * mean(x)^2) / n # The 'sample' variance formula ends with n-1 instead
}


# You can now try it out by creating some realisations of a normal distribution
z <- rnorm(1000, mean = 0 , sd = 2)
# What do you think the answer to this should be?
EmpVar(z)
```

Now on to the programming.[1]

[1] These tasks are inspired by the APTS course of Dr. R Ripley, Oxford University

In what follows you need to actually run and understand the examples given before starting the exercises.

2 R programming Tasks

In what follows, do not type in the commands that begin with `[1]`—this is simply the output you should see when you run the code.

2.1 Creating simple objects

If we simply type a command e.g.

```
1/3
[1] 0.3333333
```

we see that the output is printed directly to the screen. To create objects we use the assignment operator `<-`. If we want to print the answer then we either type the name of the object we created or place parentheses around the command e.g.

```
x <- 1/3
x
[1] 0.3333333


(x <- 1/3)
[1] 0.3333333
```

2.1.1 Variable modes

As with Python, R has several different modes of variable that it can store in memory, for example

- integer e.g. 0, 1, ... stored specifically as an integer;
- double/numeric e.g. 1.1, $\sqrt{2}$, 2, ... stored with double precision;
- character e.g. ``Treatment A'', ``Bob'' or ``Kate'';
- logical i.e. `TRUE` or `FALSE`.

Note: R will store e.g. 3 as a double unless you specifically tell it not to

You can tell R which mode you want to store a variable in you when you create it. Try creating the following and then looking at the *structure* of the variable created by typing e.g. `str(A)`. Can you recognise the output and where it tells you the mode?

```
# R will normally try and work out what mode it thinks the variable is automatically
# This can be good but it can get it wrong, in which case you need to override it
```

```
## A vector of double precision numbers
A <- c(1.2,2,3.4) # The c() command simply tacks the numbers together

## To create a vector of integers you need to force it
B <- as.integer(c(1,2,3))

## A vector of characters (needs quotes around each value)
C1 <- c("Tim", "Jane", "Kate") # R recognises that the quotes imply characters
C2 <- as.character( c("Tim", "Jane", "Kate")) # Or again you can tell it for sure

## A vector of logicals (again R recognises these automatically)
D <- c(TRUE, FALSE, TRUE)
```

2.1.2 Tasks

Now on to some exercises.

1. Create a numeric/double object called `Score` with the value 10.4.
2. Create a logical object called `isMale` with the value `TRUE`.
3. Create an object called `Name` containing the character string `Tim`.

2.2 Vectors

A vector is a collection of objects, all of the same type (e.g. double, integer, boolean, character). Vectors are extremely important in R and it treats them specially.

To create a vector we often use a simple function such as `c, rep, sample, paste` or an operator such as ‘‘`:`’’ for example

```
x <- c(0, 1.2, 8/5)
y <- 1:10
```

Important and needed for the next set of tasks:

```
t1 <- x > 2 # What does this do? What is the structure of t1? Can you see why?
t2 <- y == 4 # Similarly what does this do?
```

2.2.1 Tasks

1. R treats even single objects as a vector of length 1. This is very important as it allows lots of coding efficiencies we will describe later. To show this use the object `Score` you previously created and type `is.vector(Score)`.
2. Create a vector `height` of 10 random integers between 150 and 160, allowing repeats, using the command `sample()`. You might need to look at the help file for this command using `?sample`.

3. Using the `rep()` command, create a vector of character strings `Pest` containing 10 values with the first 3 being `"A"` and the last seven being `"B"`. Again you'll probably need to look at the help file `?rep`.

4. Using the `paste()` and the `rep()` command create a vector of character strings `Flower` containing 10 values with the first 8 being `"Helianthus debilis"` and the last two being `"Helianthus annuus"`. (Look at the `paste()` help page.)

5. Using an operator (i.e. not just entering it by hand), create a logical vector called `isTall` which has value `TRUE` if the corresponding value of `height` is 155 or more, and `FALSE` otherwise.

Note: variables that can only take a few values, such as `Pest` and `Flower` above, can be stored more efficiently in R using a more sophisticated mode of variable, a *factor*; this is particularly important for some kinds of statistical modelling. We will look at this in detail later.

Manipulating vectors

Since R considers vectors somewhat special it means you can do some clever things with them. Suppose we have three vectors (note the different lengths)

```
a <- 1:3
b <- 6:8
c <- 1:2
```

If we add/multiply vectors of the same length then they are added/multiplied elementwise e.g.

```
a + b
a * b
```

Alternatively if one vector is shorter than the other they can still be added but the shorter vector is repeated until it is sufficiently long. Note that R will give you a warning message if the number of times you need to recycle the shorter vector is not an integer.

```
a + c
a * c
a + 1
```

You can extract certain elements of vectors by using the built-in indexing. What do the following commands produce?

```
d <- 1:10
d[2]
d[-2] # Note that this is d with its 2nd element removed
d[1:9]
```

If you want you can also change elements using the same indexing e.g.

```
d[1] <- 10
d[2:5] <- c(10,12,13,14)
```

2.2.2 Tasks

1. Why is the command `a+1` an example of recycling?
2. Using your earlier objects, what will by produced by the following command?

   ```
   height[isTall]
   ```

3. Create a vector `Aheight` containing just those value in your vector `height` which have corresponding `Flower` value `"Helianthus annuus"`.
4. Do the following:
   - Create a vector `Even` of the first 100 even numbers in order.
   - By using indexing (with the `-` index), create a sub-vector `EvNoFirst` which has removed the first element of `Even` i.e. the integer 2.
   - Similarly, create a sub-vector `EvNoLast` which has removed the last element of `Even` i.e. the integer 200.
   - Calculate the difference `EvNoFirst - EvNoLast`. Is it what you expected it to be?

3 Homework

Due practical class week 2 in hard copy - i.e. bring a printed-out piece of paper containing your solutions, with your name written clearly at the top.
Your solutions must be clearly structured and be written in such a way that they are readable and understandable to the marker. Do **NOT** simply submit raw R code and output without any real world explanation. If you want to use LaTeX then you can, but you don't need to. You can submit your homework as simply a `.txt` file but it still needs to read as a proper document—split by question with explanatory text as follows.

```
Question 1

a) To create 10000 numbers from the gamma distribution with shape parameter 2
and scale parameter 4 we use code
> x <- rgamma(....)

b) We can find the mean and standard deviation using
> Some code
> Some code
```

Once the code starts getting more complicated we will also start to expect comments explaining what the code does. If you want to do some maths on a piece of paper it is fine to do this by hand and then staple it together.

Tasks

1. R has built in functions which enable us to sample from common random distributions. One such is `rgamma()` which enables us to sample from a Gamma random variable with density
$$f(x) = \frac{r^a}{\Gamma(a)} x^{(a-1)} e^{-rx}.$$
The command

```
rgamma(10, 1, 5)
```

will produce 10 random variables from the gamma distribution with shape $a = 1$ and rate $r = 5$. Do the following:

   - By looking at the help file (using the command `?rgamma`), create 10000 numbers from the gamma distribution with shape parameter 2 and *scale* 4. Store them in a vector `x`.
   - Find the mean and standard deviation of this sample.
   - Find the mean of all the entries in `x` which are strictly greater than 0.5.
   - What does the following command do?

```
sum(x > 0.5)
```

2. **Estimating** π Georges is sitting in a French cafe after his lunch break playing with his toothpick. While trying to solve a particularly difficult maths problem his toothpick falls onto his notepad on his table. His notepad happens to contain horizontal lines exactly 4 cm apart while (after much use) his toothpick is exactly 2 cm long. The toothpick happens to fall so that it crosses one of the lines in his book.

   He starts to wonder how likely it is that his toothpick would have fallen in such a way. To investigate this he repeated throws his toothpick onto his page so that the location of the centre of the toothpick has a uniform distribution anywhere on the page and the angle it makes with the vertical is also uniformly distributed.

   - Georges first works out that the distance between the center of the toothpick on one of his throws and the nearest notepad line is uniformly distributed between 0 and 2 (can you explain why?). Using `runif()` create 10000 such random distances and store them in a vector called `CentDist`.
   - By default, R uses radians to measure angles. Georges calculates that the distribution of the angles from the vertical that his toothpick falls is Uniformly distributed between 0 and $\pi/2$. Using `runif` again, create 10000 such angles and store them in a vector `Angles`.
   - We can plot an example of the toothpick and a line as shown below in Figure 1. Simple trigonometry tells us that the height d that the toothpick extends vertically above its mid-point is $\cos(\theta)$, where θ is the angle from the vertical.

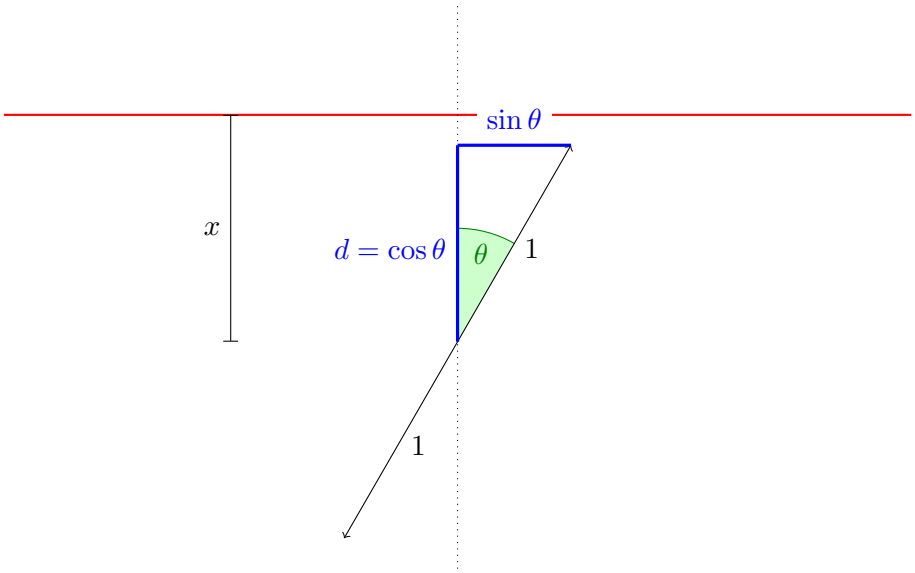

Figure 1: Plot illustrating a sample toothpick

Now if the centre of the toothpick landed x cm from nearest line then it will cross that line if $x < d = \cos(\theta)$. Transform your variable `Angles` into a vector `d` working out the vertical height that each of the sample toothpicks will extend above its centre.

- Using R calculate the proportion of Georges' 10,000 sampled toothpicks which cross lines and store it as the variable `p`.
- Calculate $1/$`p`. Does it look like a number you recognise?
- *Additional Challenge - Optional* Can you show formally that if he were to keep throwing his toothpick eventually this fraction would tend to $\frac{1}{\pi}$?

  [*Hint*: We need to work out the $P(X < f(\Theta))$ for your function $f(\cdot)$ defined earlier. The problem is that both X and Θ are random variables. Try and condition on the value of Θ using the ideas from MAS113 i.e.

$$P(X < f(\Theta)) = \int P(X < f(\Theta)|\Theta = \theta)p_\Theta(\theta)d\theta$$

  where $p_\Theta(\cdot)$ is the density of Θ.]