Assignment 1 for Statistical Computing and Empirical Mathods

Dr. Henry WJ Reeve

Teaching block 1 2021

Introduction

This document describes your first assignment for Statistical Computing and Empirical Methods (Unit EMATM0061) on the MSc in Data Science. Before starting the assignment it is recommend that you first watch video lectures 1 and 2.

1 Create your first data frame

First open RStudio.

Within RStudio create a vector called *animals* which contains the names of a few animals. What type of vector is this? Your vector might look something like this:

```
## [1] "Snake"   "Ostrich" "Cat"     "Spider"
```

Create a vector of the same length called *num_legs* which gives the number of legs of the different animals. Your second vector will look something like this:

```
## [1] 0 2 4 8
```

Now combine these two vectors in a data frame which has two coloumns - one with the names of animals, and another with their respective numbers of legs. The result should look something like this:

```
##   animals num_legs
## 1   Snake        0
## 2 Ostrich        2
## 3     Cat        4
## 4  Spider        8
```

The functions required can be found in the example below.

```
city_name <- c( "Bristol", "Manchester", "Birmingham", "London") # vector of city names
population <- c(0.5,0.5,1,9) # vector of populations
cities_populations_df <-data.frame(city_name,population) # generating data frame
```

2 Matrix operations

Use the `seq()` function to generate a sequence of numbers starting at 12 and decreasing to 2 in steps of -2. Call this vector `x_vect`. You may want to run `?seq` or `help(seq)` to help you do this. The vector `x_vect` should look like this:

```
## [1] 12 10  8  6  4  2
```

Now convert the vector `x_vect` into a matrix with 2 rows and 3 coloumns called `X` using the `matrix()` function. The result should look like this:

```
##      [,1] [,2] [,3]
## [1,]   12    8    4
## [2,]   10    6    2
```

Next create a 2 by 2 matrix called `Y` consisting of sequence of four numbers from 1-4. The matrix `Y` should look like this:

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

In addition, create another 2 by 2 matrix called `Z` which looks as follows:

```
##      [,1] [,2]
## [1,]    4    8
## [2,]    6   10
```

Recall that for 2 by 2 matrices A the transpose $A^\top$ is given by

$$A^\top = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} \qquad \text{where} \qquad A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

Use the `t()` function to compute $Y^\top$ and $Z^\top$.

Now compute the matrix sums $Y + Z$ and $Y + Z$. The result in both cases should be the same. We call such operations **commutative**. This means that reversing the order does not change the result.

Next, recall that for 2 by 2 matrices A and B, the matrix product AB is given by

$$AB = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} \qquad \text{where} \qquad A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \text{and} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}.$$

Use R to compute the matrix products YZ and ZY. Are the results the same in both cases? Is matrix multiplication commutative?

Given 2 by 2 matrices A and B, the element-wise product $A \odot B$ (also called the Hadarmard product) is given by

$$A \odot B = \begin{pmatrix} a_{11}b_{11} & b_{12}a_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{pmatrix} \qquad \text{where} \qquad A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \text{and} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}.$$

Use R to compute the element-wise matrix products $Y \odot Z$ and $Z \odot Y$. Are the results the same in both cases? Is element-wise multiplication commutative?

In addition, use R to compute the matrix product YX:

```
##      [,1] [,2] [,3]
## [1,]   42   26   10
## [2,]   64   40   16
```

What happens if you attempt to compute the matrix product XY? Explain the error message you get.

Now use compute the matrix inverse Y^{-1} via the `solve()` function:

```
solve(Y)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

Next compute $Y^{-1}Y$ and $Y^{-1}Y$ in R. Explain your result.

Now compute $Y^{-1}X$. Your result should look like this:

```
##      [,1] [,2] [,3]
## [1,]   -9   -7   -5
## [2,]    7    5    3
```

Can you do this without first computing Y^{-1}? Try running the help command on the `solve()` function by typing `?solve()` into the R console to find out how to do this.

3 Writing a simple function within R

You will now create your first R script containing a short R function. Go to *File -> New File -> R Script* to create your script. Save the file by clicking *File -> Save* and giving the file a name of your choice eg. "myFirstRScript". You can run all the code within your script by clicking on the "Source" button on the top right.

Now within your script create a short function called "myFirstRFunc" which takes in a single numerical argument n and outputs the sum of all those numbers strictly below n which are divisible by either 2 or 7 or both. For example if $n = 14$ then the list of all positive integers below n which are divisible by 2 or 7 is $2, 4, 6, 7, 8, 10, 12$ so your function applied to 14 gives the answer $2 + 4 + 6 + 7 + 8 + 10 + 12 = 49$ i.e. "myFirstFunc(14)=49". This example is adapted from Project Euler, which is a fantastic resource for algorithmic problems of varying levels of difficulty.

You may find it useful to look at the sample function within Blackboard. Make sure your function includes useful comments. You may want to include a check so that you function produces an error if it is given an argument which isn't a non-negative integer.

If you have been successful your function should produce the following output.

```
myFirstRFunc(1000)
```

```
## [1] 284787
```

4 Version control with RStudio and git

In this section we will combine RStudio with git to create an R project with version control.

4.1 Connect RStudio to git

If you do not yet have GitHub account sign up for one here https://github.com/.

Next connect your RStudio to git via the "usethis" R package. Within RStudio perform the following steps within your R console:

```
install.packages("usethis") # Install the "usethis" R package
library(usethis) # Load the "usethis" R library
use_git_config(user.name = "Bob Smith", user.email = "bob@example.org") # Set profile info
```

The first step installs the "usethis" R package. The second step loads the "usethis" R library. You may be prompted to install RTools at this stage. If so follow the links provided by RStudio. The third step sets your git profile information. The user name you enter should be the same as the username you used to set up your git profile. Similarly, the email needs to be the same email address as you used to set up your git profile.

4.2 Create an R project with version control

Go to https://github.com/ within your web browser and log into your git account. Create a new repository by clicking the green "New" button. Next:

1. Give your repository a name eg. "firstRProject".
2. Give your repository a description eg. "This is a repo for an R project".
3. Check the box which says "Add a README file".
4. Always be mindful over whether your repo is public or private.
5. Click the green "Create repository" button.

You have now created a git repository. Next click the green "Code" button and copy the url provided. This is the repository url.

Next go back to RStudio. Then go to:

```
File -> New Project... -> Version Control -> Git
```

1. Below "Repository URL:" Paste the repository url which you copied from your git repo.
2. Below "Project directory name:" Choose a directory name.
3. Below "Create project as subdirectory of:" Choose where to store your project on your local machine.
4. Check the box marked "Open in new session".
5. Click create project.

You have now created an R project with version control!

4.3 Commiting and pushing

within your new project click.

```
File -> New File -> R Script
```

Type some R code in your new R script eg.

```r
a<-1
```

Now save your new file by selecting `File -> Save` before giving a name.

Next go to the git tab at the top right of your screen. Then click the "commit" button.

On the left you will see a collection of check boxes. Here you can choose which files you want to add to your git repo. I suggest for now you check all of the files.

Write a commit message in the "Commit message" box eg. "This is my first commit". It's good practice to always include a commit message. Click the commit button. You should see a message which describes your changes. Commits act as a "snapshot" which records the current state of your repository.

You have now taken a local "snapshot" of your repository by performing a "commit". Next you want to record this snapshot on the central repository. This is done by performing a "Push". All you have to do is press the green "Push" button within the git panel in RStudio. You may be asked to enter your password. You typically only have to do this once.

Now check that your push has been successful. Go back to https://github.com/ and log into your account if you are not already logged in. Go to the repositories section and click on the name of the repository you created in the previous stage of this assignment. You should see a record of your most recent commit.

You may also want to move your previous R script entitled "myFirstRScript" into this project. You can then "commit" and "push" to upload your script to your git account.

5 Generating an R markdown notebook

Now we are going to create an R markdown notebook. You may want to do this within an R project with git version control. You can take a look at the sample R notebook described in the lectures available from within Blackboard (`Blackboard -> Resource lists -> Example R files -> Example R Markdown document`).

5.1 Creating the R markdown notebook

Let's create an R markdown notebook with html output as follows:

```
File -> New File -> R Markdown ...
```

You can then choose:

- A title for your document;
- An author name (your name);
- An output format. Let's choose HTML.

Then click "OK" to create an R markdown notebook. This will create a template project. Explore the project and note its key features:

1. There is a block of YAML at the start of the document which gives key information eg. title and output format.
2. You can create headings of sections with "#". Similarly, headings of subsections can be created with "##" and so on.
3. You can embed blocks of R code by using ` ```{r} ` before the R code and ` ``` ` afterwards.

4. By default both the code and its output are displayed. If you only want to include the output but not the code itself include the option `echo = FALSE` in the code prefix. If you don't want to include either then include the option `include = FALSE`.
5. You can include hyperlinks by writing < url-name>.

Save your R markdown notebook file as "Assignment1Notebook". You can then generate an HTML file with "Knit" button just above your scripting window.

Remove everything in your R markdown notebook except for the YAML section at the top with the title, author, date, output and the block of r code labelled "r setup". Resave your document frequently.

5.2 Inclucing blocks of code

Within your R markdown notebook a section heading called "Wave plot".

Now include a block of code via the commands ` ```{r} ` and ` ``` ` where you define a vector called x consisting of a sequence which starts at 0 and goes to 20 in increments of 0.01. You can do this using the **seq()** function.

Next create a vector called y which is of the same length of x and is such that the ith entry of y is equal to the sin function of the ith entry of x. This can be done via the R function **sin()**.

Now create a dataframe called "sin_df" with two coloumns: x and y. You can inspect the first few rows of your dataframe with the **head()** function like this:

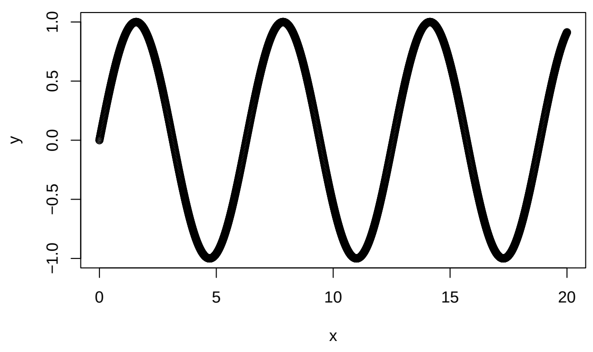
```r
head(sin_df,3)
```

```
##      x           y
## 1 0.00 0.000000000
## 2 0.01 0.009999833
## 3 0.02 0.019998667
```

If you have been successful you will observe a similar output after performing "knit".

5.3 Including plots

Now display a plot with x as the x-axis and y as the y-axis. You can do this using the **plot()** function. Your result should something like the following:



5.4 Displaying mathematical formulae

You can also use Rmarkdown notebooks to render mathematical formulae. For example, you can write `$y=\sin(x)$` inline to display "$y = \sin(x)$".

You can also display more complex expressions. For example, the code below gives rise to the following output.

```
\[ \sin(x)=\sum_{n=1}^{\infty}(-1)^{n+1}\cdot \frac{x^{2n-1}}{(2n-1)!}
\approx x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}\ldots. \]
```

$$\sin(x)=\sum_{n=1}^{\infty}(-1)^{n+1}\cdot \frac{x^{2n-1}}{(2n-1)!} \approx x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}\ldots.$$

Similarly, the code below gives rise to the following output.

```
\[ A=\left(\begin{matrix} a_{11} & a_{12}\\ a_{21} & a_{22} \end{matrix}\right).  \]
```

$$A=\left(\begin{matrix} a_{11} & a_{12}\\ a_{21} & a_{22} \end{matrix}\right).$$

For information about writing mathematical formulae in Rmarkdown documents visit here.