BRIEF INTRODUCTION TO THE STATISTICS PACKAGE R

1. Introduction

R is a freely available language and environment for statistical computing and visualisation. It is popular for data analysis, research and teaching. Courses following on from MA10212 will use R for statistical computations.

This brief introduction is intended to teach the basic points of R: starting and quitting the package, the help facilities, doing arithmetic, command structures, functions and graphics.

There are two ways to run R: the basic version of R and RStudio. The commands that you give in your code are the same in both cases. While R may be simpler to use, in RStudio it is easier to navigate directories, open and close files, etc. RStudio also has report writing features that will be used in later Statistics courses. These notes describe both R and RStudio.

Starting R or RStudio

If you wish to run R on your own laptop or PC, you must first download and install R. In order to use RStudio you must download and install both R and RStudio. See the final section of this document for more on installation.

On campus, you can use R or RStudio on a University of Bath Public Access PC.

To use a University of Bath server remotely, go to

`https://rdweb.wvd.microsoft.com/arm/webclient/index.html`

and click on the `UniDesk` icon. Once this has loaded, you should find R and RStudio in the Start menu in the usual way.

In order to access R, bring up the Start menu, scroll through the list of programs until you find “R”, left click on “R”, then left click on “Rx64 4.0.2” (or similar).

To access RStudio, bring up the Start menu, scroll through the programs until you find “RStudio”, left click on “RStudio”, then left click on the next “RStudio”.

Overview

On starting R or RStudio, you see the R console window. The simplest thing to do (which we shall discourage later) is to type commands in the console.

Experiment by typing some commands (here `>` is the prompt produced by R):

```
> 3 + 4
> sin(pi/2)
> exp( -2 )
> a <-2
> b <-2
> a+b
> c=3
> c+1
> c>4
```

R is a command line package, rather than a menu-driven one. It usually returns the prompt `>`. Generally R uses brackets at the end of a name to indicate that it is a command or function (the brackets may contain arguments for the function). For example, if you want to list the objects currently stored in R, you need to type the command `ls()` at the `>` prompt, i.e.,

```
> ls()
```

If you type `ls` without brackets, R prints the inner workings of the function `ls` — try it and see. You can use the arrow keys to scroll around in what you have typed, so if you forget the brackets use the up arrow for a second attempt.

Quitting R

When you want to stop R, type:

`> q()` *(don't forget the brackets)*

You should be asked if you want to save the workspace. If you choose "yes", a workspace file will be created in the directory you are currently working in and you can use this to re-create your R session when you start R up again.

Loading a saved workspace into R or RStudio

A workspace you have saved (from R or RStudio) is represented by an "R" icon in your directory. If you double left click on this icon, this will start an R session with the workspace loaded so, for example, any variables you defined and gave values to in your previous session will be there for you to use.

In order to use this workspace in RStudio, you need to start RStudio, navigate to the appropriate directory in the "Files" window, then click on ".RData".

2. Writing in a script

For any substantial calculation, you should write commands in a "script". There you can build up your code and test it by running all or part of it as if you had typed the commands into the console. When you see what your code produces in the console window, you can make modifications and corrections as necessary.

You can save your script to a file and upload this again in a new R session. Once you have exactly what you want in your script, run the whole script to produce a clean, complete set of results in the console. Then, you can save these results along with any figures you have drawn to put into a report.

Below, we describe the details of the above process for R and RStudio.

Scripts, etc: The R version

Scripts

To start a new script, left click on “File” and then “New script”. A window with the heading “Untitled - R editor” should appear.

You can type R commands in the script, line by line. Highlight part of the text in the script window with the left mouse button, then right click and choose “Run line or selection” (or just type Control R) to run these commands. The lines of code you have selected will run, just as if you had typed them into the R console.

You can have multiple script windows open at once. This can be useful if you want to create a set of functions in one script and write code that calls these functions in a different script.

You can save a script into your filespace. It helps if you first navigate to the appropriate directory by left clicking in the console window, left clicking on “File” and on “Change dir...”, and selecting your directory in the browser. Now left click in the script window, click on “File” and on “Save as...”, then type in the name you wish to give your saved script.

When you start a new R session and want to open a saved script, use “Change dir...” to navigate to the appropriate directory, click on “File” and on “Open script...”, then select your script and click on “Open”.

If, at any time, you wish to see which directory you are working in, the command

```
> getwd()
```

returns the name of your current working directory.

Although we have seen it is possible to save an R workspace, it is usually sufficient to save a script. Then you can start R, upload your saved script and run all the commands (e.g., by Control A, Control R) to set up the variables and functions defined in your script in the current workspace.

Saving the contents of the console window

Select “File” and “Save to file...” and specify a filename of the form yyy.txt. This will save the **commands and output** currently in the R console window, plus whatever appeared earlier in the session (which you can see by scrolling back up). You can use the contents of such a file in your solutions to Lab Sheets.

I have sometimes tried to save the console output and found I have created an empty file! To avoid this, do something in the console window, e.g., press return, just before saving. Before you close your R session, check that the newly created .txt file contains what you expect.

Saving plots

See **Graphics** and **Writing a plot directly to a file**.

Scripts, etc: The RStudio version

Scripts

After you have opened your RStudio session, you can move to the appropriate directory in your filespace. To do this, click on “Session”, “Set Working Directory” and “Choose Directory...”, then navigate to your desired directory and click on “Open”. The path to this directory and the files therein should be displayed in the “Files” window.

To start a new script, left click on the + symbol in the top left corner of RStudio, then click on “R Script”. The new script will have a name “Untitled n”. After you have typed some commands in this script, save it by clicking on the disk icon, typing a filename and clicking on “Save”. The name of the saved file should appear in the tab at the top of the script and be added to the list in the Files window.

You can type R commands in the script, line by line. Highlight part of the text in the script window with the left mouse button, then click on the green arrow to run these commands. The lines of code you have selected will run, just as if you had typed them into the R console.

You can have multiple scripts open at once. This can be useful if you want to create a set of functions in one script and write code that calls these functions in a separate script.

When you start a new RStudio session and want to use a saved script, click on “Session”, “Set Working Directory” and “Choose Directory...”, to navigate to the appropriate directory. Then click on the yellow folder icon close to the top left corner of RStudio, select and open your script. Alternatively, left click on the name of the script in the Files window.

Although we have seen it is possible to save an RStudio workspace, it is usually sufficient to save a script. Then you can start RStudio, upload your saved script and run the commands to set up the variables and functions defined in your script in the current workspace.

Saving the contents of the console window

A simple way to save output in the Console window is to highlight the text using the left mouse button, copy this by right clicking and choosing “Copy” (or type Control C). Then you can paste this text into a file, e.g., a Word document.

This cutting and pasting may seem a bit crude for a sophisticated environment like RStudio. There are more elegant methods to use, such as RMarkdown, but we shall keep things simple and not go into these methods here.

Saving plots

See **Graphics** and **Writing a plot directly to a file**.

3. Getting help

Suppose we want to draw a histogram in R. There are several sources of help.

(i) The R help function

You can type

```
> help(histogram)
```

Since `histogram` is not the relevant R command, this does not help much.

Had we known that we needed the command `hist`, we would have typed

```
> help(hist)
```

(ii) The on-line R help function

A more general approach to finding help is to use the web help facility by typing

```
> help.start()
```

If you are using R, an external browser window will open. In RStudio the browser will open in the internal "Help" window. This browser has links to lots of useful information, particularly "An introduction to R" and "Search engines & keywords": try typing *histogram* into the search engine and follow one or more of the links that appear.

(iii) Other information on the web

Type *R histogram* into Google, say, and follow some of the links. This route can often provide well written explanations that assume less technical knowledge of R than the output from commands such as `help(hist)`.

4. Using R as a calculator

R can be used as a calculator. Try

```
> 12 - 4
> cos( 0 )
> tan( pi/4)
> tan( pi/2)
> log(10)
```

None of the above commands stores the result in a variable. To assign values to an object, use either `=` or `<-`. To see the value of an object, type its name:

```
> temp2 <- 3**2
> temp2
> aa = temp2 + temp2
> aa
```

Vector operations

Since data are often stored as vectors or matrices, R has a large number of functions to deal with these types of object. To illustrate vector arithmetic, we shall consider a Body Mass Index example from the book "Introductory Statistics with R" by P. Dalgaard.

R uses the notation `c()` to denote a vector, so to enter 6 weight measurements into a vector called `weight`, type

`> weight <- c(60, 72, 57, 90, 95, 72)` *(the spaces are optional — but the commas are essential)*

To carry out some further calculations, add

```
> height = c(1.75,1.80,1.65,1.90,1.74,1.91)
```

R applies arithmetic operations element by element, so in order to calculate Body Mass Index, $weight/height^2$, for these 6 individuals we can type

```
> bmi = weight / height ^ 2
> bmi
```

We can also perform various selection operations, for example

`> height[1]` to find the height of the first person
`> height[3:5]` — here, `3:5` is shorthand for the sequence `3,4,5`.
`> length(height)` to find the number of elements of height
`> sum(height)` to find the sum of the heights
`> sum(height)/length(height)` to find the average height
`> mean(height)` a quicker way to find the average height
`> max(weight)` to find the heaviest weight
`> range(bmi)` to find the range of body mass indices

We can introduce logical operators and selection by logical conditions:

`> bmi > 25` returns a vector of true and false values corresponding to whether the matching element of bmi is over 25 or not

`> weight[bmi<20]` gives weights of those people with bmi less than 20
`> sort(bmi)` sorts the values in `bmi` from smallest to largest

`> order(bmi)` tells you the positions in which the ordered values of bmi, from smallest to largest, appear in the vector `bmi`

`> height[order(bmi)]` arranges the values in height in the order corresponding to increasing values of `bmi`

A common mistake is to try to use `height(1)` — instead of `height[1]` — to refer to the first element of the vector. This causes R to try to interpret `height` as a function, rather than a vector, and results in an error.

5. Graphics

Calling a plotting command in R opens a graphics window in the top right hand corner of the screen. In RStudio, the plot appears in the “Plots” window.

The command

`> plot(height)` will produce a plot of the 6 values of height (on the y-axis) against the order they occur (on the x-axis).

In R you can move plot windows or re-size them using the left mouse button on the window edge in the usual way.

In RStudio you can change the size of the Plots window and, hence, change the aspect ratio of a plot. Before you draw a plot, make sure the Plots window is reasonably large — if it is too small you will get the error message “figure margins too large” when you try to draw a plot.

In RStudio, clicking on the arrows at the top of the Plots window will move through the current set of plots.

It is worth discovering some of the other arguments for the function `plot()`:

```
> help(plot)
> plot(height,type="l",xlab="Order",ylab="Height",main="My plot")
```

To plot one object against another, give two arguments:

```
> plot(height,weight,xlab="Height",ylab="Weight",main="H-W plot")
```

Note which variable is plotted on the x-axis and which on the y-axis.

In R you can save a graph to a file by right clicking on the graphics window. Then, choosing “Save as metafile” will create a file with ending .emf in the directory that you specify, while “Save as postscript” will create a file with ending .eps. To view the figure later, left click on the icon for the .ems file or the .eps file and you should be able to see the content of your file using Paint or GSview, respectively.

In RStudio, you can click on “Export” and “Save as Image”, then choose from a selection of image formats.

In R, by default, each graphics command overwrites what is in the current graphics window. To avoid this, you can create a new window by typing

```
> windows()
```

The old graphics window will still be there — just move the new window to see the old one underneath.

Another useful option in both R and RStudio is to set the parameter `mfrow()` in order to put multiple plots on one page. Once you have set `mfrow()` using the `par` command, plots will continue to be placed in the specified format.

`> par( mfrow=c(2,2) )` produces 2 rows each of 2 images
`> par( mfrow=c(2,1) )` produces 2 rows each of 1 image
`> par( mfrow=c(1,1) )` returns the set-up to 1 image

Writing a plot directly to a file

Another option is to type commands to write a plot directly to a file. Various file types are possible. Suppose, for example, you wish to save a plot of weight against height to a .png file called plot1.png. This can be achieved by typing

```
> png(filename="plot1.png")
> plot(height,weight,xlab="Height",ylab="Weight",main="H-W plot")
> dev.off()
```

In the above sequence, the first command sets up the file plot1.png as the destination for the graphic. The second command creates the plot — which should look just the same as the one previously produced on the screen. The third command ends this process, so further graphics will appear in a graphics window on the screen in the usual way.

When creating a plot this way, the file takes over the role of the window you would otherwise see in your R session. Thus, it can contain just one plot, or an array of plots if the `mfrow()` command is used.

I have had problems viewing .eps files or importing them to Word documents. While an .ems file works better, this tends to contain a small graphic and a lot of empty space that can be hard to remove. Using a .png file appears to be a safe option with robust functionality.

6. Combining material in your coursework solutions

For your coursework, you will combine the transcript of your R session and the resulting graphics output into a Word document. To do this, open a Word document, then cut and paste content from the .txt file containing your R session (see "Saving the contents of the console window"). You can insert the .emf, .eps or .png files for graphics showing histograms, plots, etc. To do this in Word, choose "insert", then "picture", then navigate to the .emf, .eps or .png file and load this.

Another option is to copy the graphic directly from the R graphics window then and paste this item into your Word document. In R, you can copy the graphic by right clicking and choosing "Copy as bitmap". In RStudio, click on "Export" and "Copy to Clipboard...".

Place each graphic after the R commands that produced it. Be sure to comment on what R has produced so as to answer the specific questions asked in the Lab Sheet. Writing these answers in colour helps highlight them for the marker.

For a really good piece of coursework, tidy up the output: remove the first part of the console output that tells you R is starting up, and remove any mis-typed commands and their output. Add titles and axis label to your graphics. Where appropriate, re-size the graphics in the Word document to make this more readable. You will then upload the finished Word document to Moodle as your submitted work.

Important:

Submit commands and their output from the console window.

Submit your R code, not a picture of it.

Tutors marking your work need to see both your code and the output it produced

Sometimes, we wish to run your code ourselves to understand what it is doing or to track down a bug. For this, it helps to have a text version that we can cut and paste into an R session. If you take a snapshot of your code and paste this into your submission, all we have is a picture of your code: if we want to see what it produces, we have to type it all into R ourselves.

7. Flow control

A strength of R is that you can write your own functions. Before considering an example, we need to know the flow control arguments that are available to define *for loops*, *while loops* and *if-else statements*.

You should experiment by trying out some of the following examples. However, it is not possible to correct errors after typing a number of lines in the console window. So, before trying examples, you may find it helpful to re-read the earlier section on **Scripts**.

The `for` loop:

Suppose we want to calculate $\sum_{i=1}^{100} i^2$. One way to do this in R is to use a `for` loop, putting the result in the variable `sum100`:

```
> sum100 <- 0
> for ( i in 1:100 )
+  {
+   sum100 <- sum100 + i**2
+  }
> sum100
```

This code sets `sum100` to zero, then carries out a loop for `i` in $\{1, \dots, 100\}$. The curly brackets { } contain the commands to be carried out for each value of `i`. Note the R prompt changes from `>` to `+` when the current command has not yet finished. In this case, the `for` loop must be terminated by a final "}".

In step `i`, `sum100` is incremented by `i**2`, and the loop continues to the next value of `i`. At the end the value of `sum100` is printed.

The `while` loop:

We can use a `while` loop to find the smallest `n` for which $\sum_{i=1}^{n} i^2 > 100$.

```
> n = 0
> sum100 = 0
> while ( sum100 <= 100 )
```

```
+  {
+    n = n+1
+    sum100 = sum100 + n**2
+  }
> n
```

This code initialises the variables `n` to 0 and `sum100` to 0. Then, while the current value of `sum100` is not more than 100, it first increments `n` by 1 and then increments `sum100` by `n**2` (using the increased `n`). Eventually the test condition is no longer true and the terms within { } are not carried out. The final value of `n` is printed — and this is the value we wanted to find.

The `if` construction:

The general form of this construction is

```
> if (test condition)
+  {
+    some commands
+  }
```

If the test condition is TRUE, the commands are executed. If it is FALSE, they are ignored.

Examples of test conditions are `x > y`, `x <= y`, `a==b` and `s != t`. The last two test whether `a` is equal to `b` and whether `s` is NOT equal to `t`

The `if, else` construction:

The general form of this construction is

```
> if ( test condition1 )
+  {
+    some commands1
+  } else if (test condition2 )
+  {
+    some commands2
+  } else
+  {
+    some commands3
+  }
```

This structure checks to see if `test condition1` is TRUE and, if so, executes `some commands1`. If the first condition is FALSE, it continues on to the other sections: then if `test condition2` is TRUE, it executes `some commands2`, but if `test condition2` is FALSE it executes `some commands3`.

You can insert more cases of `else if`. Alternatively, the `else if` part can be omitted altogether, giving a simple `if / else` command.

It is important that `else if (test condition2)` and `else` are on the same line as the preceding }. If the } were on a line by itself, this would finish the `if` statement, thus R executes the completed command, returning the `>` prompt, and the following `else` or `else if` command causes an error message.

We illustrate `if` and `else` structures in the next section, **Writing R functions**.

There are a number of other useful flow control commands in R. Check them using `help(Control)` — with a capital C.

8. Writing R functions

Suppose you want to repeat a set of commands, possibly using different parameter values. Rather than type all the code each time, you can define a function to carry out the calculations, passing parameter values to the function as arguments.

We illustrate this idea with a simple function that generates and displays binomial data. Type the following commands

```
> show.binomial = function(size,n,p)
+  {
+   sample = rbinom(size,n,p)
+   # Draw a histogram of the data in sample --- this is a comment
+   hist(sample)
+  }
```

You have just defined a function called `show.binomial` which takes three parameter values. Implement the function by the command

```
> show.binomial(100,20,0.5)
```

In response, R generates a sample of 100 independent Binomial$(20, 0.5)$ random variables (RVs) using the built in function `rbinom()` and draws a histogram of the 100 values. Repeat the command and experiment with other parameter values.

The following, more substantial function can be used to demonstrate that the sum of two independent binomial RVs with the same probability p is also binomial.

```
add.binom = function(nreps,n1,n2,p)
 {
  sample1 <- rbinom(nreps,n1,p)
  sample2 <- rbinom(nreps,n2,p)
  sample3 <- sample1 + sample2
  par(mfrow = c(3,1))
  if (nreps > 50)
  {
    hist(sample1, xlim=c(0,n1))
    hist(sample2, xlim=c(0,n2))
    hist(sample3, xlim=c(0,n1+n2))
  } else
  {
    stripchart(sample1, method="stack", xlim=c(0,n1),
    ylim=c(0,nreps/2))
    stripchart(sample2, method="stack", xlim=c(0,n2),
    ylim=c(0,nreps/2))
    stripchart(sample3, method="stack", xlim=c(0,n1+n2),
    ylim=c(0,nreps/2))
  }
 }
```

Writing a function in a script

It is easier to modify and correct a new function if you write it outside the R console. Open up a script open in which to put functions. Then, rather than type in all the commands for the function `add.binom`, go to the course Moodle page, click on the link to "Script for add.binom and add.binom2" under **R sessions and R code**, and cut and paste the text into your script.

Highlight and run all of the text for the function `add.binom`. The effect is just as if you had typed all this text into the R console. So, you have created the function `add.binom`. Implement it with some parameter values, e.g.,

```
> add.binom(100,20,20,0.4)
```

You should get an error message! In fact, there are two errors in the script from the Moodle page but only one will have caused a problem this time. When you have identified the error, edit the text in the script window, run these commands again to re-define the function `add.binom`, and repeat the `add.binom` command.

The above function generates a vector of `nreps` samples from a Binomial$(\mathrm{n1}, \mathrm{p})$ distribution and another vector of `nreps` samples from a Binomial$(\mathrm{n2}, \mathrm{p})$ distribution. The vectors are added element-wise to give `nreps` realisations of a Binomial$(\mathrm{n1}+\mathrm{n2}, \mathrm{p})$ RV. The function plots the two sets of random samples, together with their sum. If the sample size `nreps` is large, the plots are histograms, otherwise, dotplots are used. You can use the help facility to see what the arguments to these functions mean.

It is often desirable that a function generates output to be used later. We do this by giving the `return` command just before closing the function. Try reading in the following function `add.binom2` from "Script for add.binom and add.binom2".

```
add.binom2 = function(nreps,n1,n2,p)
 {
  sample1 <- rbinom(nreps,n1,p)
  sample2 <- rbinom(nreps,n2,p)
  sample3 <- sample1 + sample2
  par(mfrow = c(3,1))
  if (nreps > 50)
  {
    hist(sample1, xlim=c(0,n1))
    hist(sample2, xlim=c(0,n2))
    hist(sample3, xlim=c(0,n1+n2))
  } else
  {
    stripchart(sample1, method="stack", xlim=c(0,n1),
    ylim=c(0,nreps/2))
    stripchart(sample2, method="stack", xlim=c(0,n2),
    ylim=c(0,nreps/2))
    stripchart(sample3, method="stack", xlim=c(0,n1+n2),
    ylim=c(0,nreps/2))
  }
  outvar=sum(sample1)
  return(outvar)  # The function will return the value
                  # of outvar as its output
 }
```

Then type

```
> add.binom2(100,10,10,0.2)
```

and also try

```
> ww=add.binom2(100,10,10,0.2)
```

followed by

```
> ww
```

Tidying up

You will not necessarily want to save all the objects you create during an R session. Use `rm()` to remove them.

```
> rm(height, weight, bmi)
```

Also, if you have finished writing a script and want to check that it does what it is meant to do, it is good to run functions and commands in a clean workspace. That way, you can be confident that the variables are being defined as intended by commands in your script and you are not just using variables that happen to exist already in your R workspace.

You can empty your workspace with the command

```
> rm(list=ls())
```

— but it goes without saying that you should use this command carefully.

9. Installing R and RStudio on your own computer.

See `http://www.r-project.org`, if you wish to install R on your own computer (free of charge). On a UNIX or MAC platform, the software has a somewhat different appearance from that seen under Windows but its functionality will be as described here.

If you wish to run R through RStudio first install R, then download and install RStudio from `https://rstudio.com/products/rstudio/download/`. Note that RStudio is an environment in which you run R, so you do need to install R first so that RStudio can use it.

CJ, February 2022