Introduction to R

This document contains modules to introduce R and is available online from

`http://www1.maths.leeds.ac.uk/~arief/R/intro/`.

Each module gives some examples of R codes or commands, starting from the very basic level. The idea of this course is to put emphasis on "hands-on learning" to understand R . Run the R commands in the examples *one line at a time* and understand what is going on in each line. Where appropriate, there will be some exercises that you need to answer at the end of each module.

Module 1: Installation of R

1. Open the R project webpage. Locate the link for CRAN (Complete R Archive Network).
2. Continue until you find the link to download the R executable file (under Windows).
3. Open R .
4. Quit R by selecting the option from the pulldown menu. Select *No* to the question of saving the image (we will discuss this later).
5. Open R again, and quit again using the command line `q()`.

Module 2: R window features

1. Open R , and change the working directory to your own directory.
2. Open an R script editor.
3. Open an HTML help search engine. Try to search the help page for `mean`.

Module 3: Basic arithmetic operations

In this module, we will learn how to perform basic arithmetic operations in R . When we perform any analysis in R , we create R objects, that can manifest in different forms (we cover majority of them in this course). These R objects have some names that we can create for them. However, R has some predefined functions, and it is recommended not to name our R object after existing function names. For example, the letters `c`, `t`, `q`, `T`, and `F`, are standard function/object names in R . To check whether a particular name has been taken as a function name, type the name in R and press `[Enter]`. If an error message appears saying that the object is not found, then we can use that name. For example, if you type `a` at the prompt '>', you will get the following output:

```
> a
Error: object 'a' not found
```

which means that there is currently no object in R that with the name `a`. Later, if we define an object with the name `a`, then it will produce some output when you call it as above. If you type some names that are the names of built-in functions in R , then you will have the following output:

```
> c
function (...)  .Primitive("c")
```

which indicates that the name is already taken.

Note: Two more features that we need to understand before we start are '#' and assignment operator. Anything that goes to the right hand side of '#' are considered as comments (they are ignored, or not executed by R). The assignment operators in R are '=' or '<-'. Both assign the value on the right side of the operator to the object on the left side.

Perform the following operations in R by running the codes *one line at a time* and understand what each line means and/or what are the outcomes (including if there is an error message).

Example:

```
a = 3
a

a <- 5
a

a < - 5  # Note there is a space between < and -

b = 4
a = b = 7
a
b # Is a equal to b?
```

Example:

```
a = 3
b = 5
a+3   # Addition
b-4   # Subtraction
a*2   # Multiplication
b/2   # Division

d = b+4 # Addition, and the result is put in a new object called d
d       # To check the content of d

k = d/a # Can you guess what is the value of k and m?
m = d*a

10 -> w # Assignment to the right, not recommended
w
w -> 10 # Why the error message?

w/a/b   # R executes from left to right
w-a*2   # R prioritises * and / over both  + and -
a**2    # What happens here?
a^2     # Should be the same as a**2
a^^2   # Why the error message?
```

Example:

```
?sqrt     # Get the help page for the function sqrt()
sqrt(9)   # What does sqrt() do?
sqrt(16)
sqrt(25)
sqrt(a^2) # What happens here?
a


exp(0)   # e-based power (natural number)
exp(1)
log(1)
log10(1)
log(exp(2)) # What is the base here?
log10(exp(1)) # What is the difference
              # with the previous line?
log5(exp(1)) # Does this work?

?log # Type this to get to the help page and find out

abs(-10)
abs(20)
abs(w-a*2) # What does abs() do? Get the help page

7 %/% 3
7 %% 3
8 %/% 3
8 %% 3
9 %/% 3
9 %% 3
w %/% a
w %% a    # What do %/% and %% do respectively?
```

Module 4: Exercises

First, without running the following R commands, can you guess what is the output of each question? You may then run the commands to confirm your answer.

1. ```
   g=10
   h=i=j=k=g
   h
   i
   j
   ```
2. ```
   a = 10
   sqrt(log(10*(1/a)))
   ```
3. ```
   g=10
   a=25
   exp(log10(a*4/g/g))
   ```
4. ```
   g=10
   a=25
   sqrt(a%%g*(a*2/g))
   ```

Module 5: Working directory

Note: R can facilitate the menu options from its command line. For example, to list all of the objects ('variables'), we can use the command

```
ls() # This command lists all the objects in the global workspace
?ls  # Get the help page
```

and to change (or show) the working directory we can use the commands

```
# change the working directory to 'your-destination-directory'
setwd("your-destination-directory")
getwd() # get the working directory

?setwd  # Get the help page
?getwd
```

Another example, if we wish to remove all objects in our R session (there is an option on this from the pull-down menu), we can use the command

```
rm(list=ls(all=T)) # remove all objects
rm(a) # remove the object a
```

Module 6: Logic

Perform the following operations in R related to logical TRUE/FALSE by running the codes *one line at a time* and understand what each line means and/or what are the outcomes (including if there is an error message).

Example

```
a = 5
b =10
d = -5

d < b
d < b < a # Why is the error message?
(d < b) & (b < a)
(d < b) | (b < a)
(d < b) & (b < a) | (d < a)
(d < b) | (b < a) & (d < a)
((d < b) | (b < a)) & (d > a) # Did you get FALSE?
                              # Is that what you expect?

a < abs(d)
a <= abs(d)

a == d # What is the difference between '=' and '=='?
a != d # What does '!' represent?
a != abs(d)

?'=='  # Get the help page
?'|'   # Get the help page

old.a = a # What is the purpose of this line?
a == d
```

```
a = d
a
d
old.a
```

Module 7: Logic - ‘special values’

Perform the following operations in R related to ‘special’ values by running the codes *one line at a time* and understand what each line means and/or what are the outcomes (including if there is an error message).

Example:

```
a = NA # NA represent not available
is.na(a)

b = log(0)
is.finite(b) # This and the next command check if
             # the element is finite or infinite
is.infinite(b)

d = 10/0
is.finite(d) # Do you expect d to be finite?
is.infinite(d)
```

Example:

```
exp(-Inf) # Inf and -Inf are positive and negative infinity
Inf*0 # What output do you get? What does it mean?
0/0
0/Inf
NA+2
NA/NA
```

Module 8: Exercises

First, without running the following R commands, can you guess what is the output of each question? You may then run the commands to confirm your answer.

1. `is.numeric("5")`
2. `is.logical(5>6)`
3.
```
a = 9
sqrt(a) > pi
sqrt(a)*pi <  a*pi
```
4.
```
b = 16
sqrt(b) * sqrt(b) == b  # Is this what you expected?
isTRUE(all.equal(sqrt(b) * sqrt(b), b))
```

Module 9: Construction of vectors and its basic manipulation

R works best with vectors. R applies a function ***element-wise*** to a vector. Here are some instructions to work on the R codes.

Example:

```
a <- c(1,2,3,4,5) # c is a built-in function in R, so
                  # don't use it to name an object
                  # or a variable

                  # The function c combines values into
                  # a vector or a list
a

b <- 1:5
b               # Are they the same?
?c # Get the help page
```

Example:

```
# Run the following function scan() and press [Enter].
# When prompted, start putting in the values 1, 2, ..., 5
# and followed by [Enter] each time.
# After the last entry, press [Enter] again.

d <- scan()

g <- seq(0,10,by=2) # What do the commands ‘seq’
                    # and ‘by’ do respectively?
g
summary(g)

h <- seq(0,100, length=21) # What does the command ‘length’ do?
h
length(h)

h2 <- sample(h)
h2
h3 <- sample(h, 5)
h3

?sample # To find out how to use random samples and permutations

sort(h3)
sort(h3, decreasing=T)

order(h3) # What do you think they are?
          # Get the help page!
```

Example:

```
a + 1
b*2
d^2
a^2/2
a+g # Why the warning message?

mean(log(g)) # Why?

a%in%g # What is the length of the output?
       # What is the length of each a and g?
```

```
g%in%a # What do you think is the output?

# Therefore, what does %in% do?
```

Module 10: Character and logical vectors

Run the following R commands and understand what is going on in each line.

Example:

```
x <- c("a", "b", "c", "d")
y <- c("A", "B", "C", "D", "E")

x == c("b")
x == c("m")
x == c("b", "m")
y >= "B"
y < "Z"

z <- as.character(1:5)
z
z+10 # Why the error message?

v <- c("A", "b", "C", "d")
x%in%v
x%in%y
v%in%x
v%in%y

condition = y > "A" & y < "E"
condition

y[condition] # the '[]' access the entries of
             # y based on the condition
y[y > "A" & y < "E"] # directly putting the condition

y[c(2,3,4)]
y[2:4]
x[condition]
```

```
letters
letters[1:5]
LETTERS[1:5]
```

Example:

```
#revisit this again:
sort(h3)
sort(h3, decreasing=T)

order(h3) # what do you think they are?
order(h3, decreasing=T)

h3[order(h3)] # what happen here?
h3[order(h3, decreasing=T)]
```

Module 11: Exercises

Based on the above examples, answer the following questions:

1. Create a vector called `sales` where the entries are 21, 13, 18, 19, and 11. Write down an R command that will extract the sales
   (a) above 15,
   (b) more than 14 but less than 20,
   (c) less than 16.
2. The above sales figures come from stores in the following cities (respectively): Leeds, Bradford, Manchester, York, and Hull. Put the cities in an R vector called `cities`. Using R commands, show the cities that have sales
   (a) above 15,
   (b) less than 19, and
   (c) more than 12 but less than 20.
3. Create a vector x whose entries are from 0 to 10. Create a vector y whose entries are the square of x. Write down an R command that will extract
   (a) the values of x which correspond to y above 50 and

(b) the values of y which correspond to the values of x less than eight but more than one.

Module 12: Construction of matrix

Run the following R commands and understand what is going on in each line.

```
a <- 51:65
b <- matrix(a,5) # What does the 5 do?
                 # What size is the matrix b?
br <- matrix(a,5,byrow=T) # What does 'byrow=T' do?
b
br


length(b)
dim(b)
nrow(b)
ncol(b)


# Accessing the matrix based on rows and columns
b[3,2]
b[1:3,1:2]
b[1:3,] # Mind the position of comma
        # What happened when you leave it blank after the comma?
b[,1:2] # Mind the position of comma

d <- letters[1:5]
d=="c"
b[d=="c",] # Which rows from matrix b are displayed here?
b[d<"d",]
b[d>"b",]

# vectorise b and br
c(b) # Or as.vector(b)
     # This turns matrix b back to vector a
c(br) # Or as.vector(br)
      # This is different from vector a
```

```
h <- 1:12
h
mh <- matrix(h, 5) # Why the warning message?
mh # Is the matrix what you have expected?
```

Module 13: Construction of matrix - binding vectors

A matrix can also be constructed by binding some vectors. There are two functions involved: `cbind` ('column bind') and `rbind` ('row bind'). Run and understand the following R commands.

Example:

```
a <- 1:5
b <- 11:15
d <- 21:25
f <- 31:34 # note the length of f is different to the others

cbind(a,b)
cbind(b,d)
cbind(a,b,d)
cbind(cbind(a,b),d)

rbind(a,b)
rbind(a,b,d)
rbind(rbind(a,b),d)

rbind(matrix(a),matrix(b)) # What does the command 'matrix'
                           # do to the vector a & vector b?

cbind(a,d,f) # Why the warning message?

f[5] <- 35 # What does this mean?
f
length(f)

cbind(a,d,f) # Does it work now?
```

Module 14: Basic matrix manipulation

Matrices are very important in statistics. R considers matrix as two dimensional 'vector'. The use of the term 'vector' here may not be conceptually correct (the term 'array' is more suitable). However, since R was designed to mainly handle vectors, R technically works with matrix on a 'vector-by-vector' basis. So, run the following R commands and understand what is going on in each line. Consult the help page for more information. These are just some basic manipulation on matrices.

Example:

```
a <- matrix(11:20,5)
a

t(a) # Transpose a matrix
     # Note: do not name your object as 't' as it is a
     # built-in function in R to transpose a matrix
```

Example:

```
# Basic arithmetic
a+5
a-10
a*2
a/2
a^2
# Note the above operations apply element-wise

# Some basic matrix arithmetic
b <- matrix(1:4,2)
b
d <- matrix(1:25,5)
d


# matrix multiplication
a %*% b
a %*% d     # Why the error message?
t(a) %*% d  # Why does it work now?
```

Module 15: Construction of data frame and its basic manipulation

Run the following R commands and understand what is going on in each line.

Example:

```
x = c("Focus", "Prius", "Civic", "Astra", "Corolla")
y = c(30.2, 48.4, 34.9, 37.5, 31.6)

car.data <- data.frame(x, y)
car.data

names(car.data)
names(car.data) <- c("car", "mpg") # What is going on here?
car.data

car.data2 <- data.frame(car=x, mpg=y)
car.data2  # Check the column names

# Are car.data and car.data2 the same?

car.data[,1]
car.data[,2]
car.data[2:4,]

car.data$car
car.data[1] # Did you get the same output as the command above?

car.data$mpg
```

Module 16: Exercises

Based on the above examples, answer the following questions:

1. Consider a vector called `price` whose entries are: 22, 32 ,23, 24, 23. These are the prices of cars (respectively, in thousands US$) in the above example. Construct a data frame on the (type of) cars and their price. Construct a data frame (`car.data`) where you have all three columns of information: cars, mpg, and price.

2. Doe has been quite fortunate to have US$ 19k in his bank account, and he is interested to buy a car listed in the data frame `car.data`. Construct a vector called `difference` whose entries are the difference between the car price and his bank balance. Incorporate this vector in the data frame `car.data`. Doe is prepared to ask for additional loan of no more than US$10k to cover the difference, and is interested to get a car that can go for more than 32 miles per gallon. Using `R` commands, show the observations (rows of `car.data`) that meet his criteria.

3. Construct a matrix called `x` of size 5×2, where the entries are one to ten, with one to five in the first column and six to ten in the second column from top to bottom. Construct a matrix, called `y`, whose entries are the square of the entries in `x`. Replace the third row of matrix `x` with the entries: 13 and 18. Show the entries of `x` which are less than or equal to ten.

4. Transpose the matrix `x`.

5. Construct a positive definite matrix, called `xxt` as a result of matrix multiplication of the transposed `x` and `x`.

6. What is the determinant of `xxt`? Extract the diagonal entries of `xxt`.

7. Construct a diagonal matrix of size 10×10 whose diagonal entries are integers from 11 to 20. Divide the matrix by 10. Find the trace of the the new matrix.

Module 17: Construction of list and its basic manipulation

In R , *list* is just like a clothesline where you can 'hang' different type of clothes or fabrics after you washed them. So, *list* can contain different types of R objects, e.g. data frame, matrix, vectors (numeric, characters, logicals), and another list. The components of lists can have names or not. If a component has a name, then it can be accessed by the $ operator after the name of the list. If a component has no name then it can be accessed by the '[[i]]' where i is the order of that component in the list.

Run the following R commands one line at a time and understand what is going on in each line.

Example:

```
L1 <- list(1:3, c(31:40)<36, matrix(41:44,2), LETTERS[1:6])
L1

L1[[1]]
L1[[2]]
L1[[3]]

names(L1) # What is the output here?

names(L1) <- c("n", "condition", "trans.matrix", "groups")
L1
names(L1)
```

Example:

```
a <- sample(1:100,10)
b <- a>=50
d <- matrix(c(13,52,44,29),2)
k <- letters[15:18]
p <- list(seq(100,500,100), matrix(1:4,4), 2, "slow")

a;b;d;k;p
```

Example:

```
L2 <- list(myvector=a, mylogic=b, mymatrix=d, mychars=k,
           inside.list=p)
           # This creates a list inside a list
L2

names(L2)
L2[[3]] # Accessing the third component on the list
L2$mymatrix # Accessing the same component by its name

L2[[5]]
L2$inside.list
L2$inside.list[[1]] # What are you accessing?
L2$inside.list[[1]][3] # Note about the brackets
L2$inside.list[[1:2]] # What output did you get?

matrix(c(L2$inside.list[[2]]),2) # Remember that the command c
                                 # vectorise L2$inside.list[[2]]
```

Example:

```
L3 <- list(sales=c(13,12,42,12), growth=c(5.6, 2.3, 6.7, 1.2),
           cost=c(12,15,NA, 11))

lapply(L3,summary) # What does lapply do?
lapply(L3,mean) # Why the NA?
lapply(L3,mean,na.rm=T)

sapply(L3,mean) # What's the difference with lapply?
sapply(L3,mean,na.rm=T)
```

Example:

```
# adding component to list and combining list
List1 <- list(a=1:10, b=letters[1:5])
matrix1 <- matrix(1:4,2) # add this matrix to list1
List1.new <- c(List1, list(matrix1))

List2 <- list(names=c("Joe", "Jack", "James"), jobs=c("Policeman",
            "Fireman", "Paramedic"))

List1and2 <- c(List1, List2) # combine
```

Example:

```
# Input a list to change a component
# Suppose James is an accountant and not a paramedic.
# How do we change this?

List2.old <- List2
List2$jobs
List2$jobs[3] <- "Accountant"
List2.old
List2
```

Module 18: Exercises

Answer the following questions.

1. Construct the following R objects as data from a company:

   (a) a vector with numeric entries: 100, 150, 200, 250, 350, using the command `seq()`. Note that the gap in value in the last two values is different to the other gaps of value in the sequence.

   (b) a vector with character entries: Leeds, Manchester, Hull, York, and Sheffield.

   (c) a data frame where the first column has the entries 2001, 2002, 2003, and 2004, and the second column has the entries 5.2, 3.7, 4.9, and 4.1. The name of the first column should be 'Year' and the second column be 'Percentage'.

   Create a list (called 'mylist') where the components are the three R objects with the names 'Turnover', 'Cities', and 'Increase'.

2. On the list `mylist`, perform the following tasks:

   (a) Show the cities whose turnover are at least 200.

   (b) Show the turnover of the cities: Leeds, Manchester, and Sheffield.

   (c) Show increase of the company's turnover over the years. Show the increase in decreasing order of year (so that the first row is 2004, the second row is 2003, and so forth).

3. Suppose the company has some data on cost as 50, 80, 140, 190, and 270. Put those values into a vector and add this vector to `mylist`.

4. Suppose the company has data for market share as 10, 12, 15, and 16 percent for the year 2001 to 2004, respectively. Put this information in the data frame 'Increase' in the list `mylist` as an extra column.

5. It turned out there is an error in the data. Change the percentage increase in `mylist` for the year 2003 from 4.9 to 4.6.

Module 19: If-statements

If-statements can be done in R Using the conditional `if() ... else ...` or `ifelse()`. In using the command `ifelse()`, the function will evaluate the first argument of the function. If that argument is true, the second argument will be returned. Otherwise, the last argument will be returned. See the help page for this function. Run the following R codes and understand what is going on in each step.

Example:

```
if(TRUE) print("Statement is TRUE") else print("Statement is FALSE")

if(100>200) print("Statement is TRUE") else print("Statement is FALSE")

ifelse(is.na(NA), "Statement is TRUE", "Statement is FALSE")

ifelse(is.infinite(0), "Statement is TRUE", "Statement is FALSE")

mycities <- c("Leeds", "Manchester", "Bradford", "York")
mychoice <- "Leeds"

# cat: to print, \n=" go to new line"

cat("I have chosen",mychoice,"\n")

if(mychoice %in% mycities){
cat(mychoice,"is in my list of cities \n")
 } else {
  cat(mychoice,"is NOT in my list of cities \n")
 }

# Note, if you use {} brackets in if .. else ..,
# then the word ‘else’ must be in the same line
# as the closing bracket of ‘if’ and opening bracket of ‘else’.

# Run the above if .. else .. when you change your
# choice to "London" or "Newcastle".
```

Module 20: Input and Output

The input and output are handled by some (paired) commands. They are paired based on the type of information that are read or saved.

Type of data	Reading from file	Saving to file

R object	`load()`	`save()`
		`save.image()`
		(for **all** objects)

Table	`read.table()`	`write.table()`
CSV	`read.csv()`	`write.csv()`

Run the following R codes and understand what each line does. Before running them, check that you have the desired working directory.

Example:

```
a <- LETTERS[1:10];a
x <- 1:100;x
y <- data.frame(matrix(x,10));y
ls()
save.image("all.RData")
# Quit R, don't save the workspace.

# Open R again, set the working directory
load("all.RData")
ls() # all should be there: a, x, and y

# For txt files
write.table(y, file="table-y.txt") # table format
save(y, file="y-only.RData") # R data format
# Quit R, don't save the workspace

# Open R again, set the working directory
load("y-only.RData")
ls()
y
y.tab <- read.table("table-y.txt")
y.tab
```

```
# Similarly for csv files
write.csv(y, file="table-y.csv") # csv format
y.csv <- read.csv("table-y.csv")
y
y.csv
# Open the files table-y.txt and table-y.csv in your
# text editor and see the difference.
```

Module 21: Plotting - basic plots

To draw a figure (plot), R uses the command `plot` for basic plots. Read the help page by typing `?plot.default` (and press [Enter]) at the prompt. Run the following command and understand what is going on in *each* line.

Example:

```
y <- 101:110
plot(y) # call this figure (1)

x <- 201:210
plot(x,y) # call this figure (2)
# What is the difference between
# figure (1) and (2)?

x <- 11:20
y <- rnorm(10) # What does rnorm do?
plot(x,y)
plot(x,y,type="l") # What does "l" represent?
plot(x,y,type="h") # And "h"?

?plot # If you want to find out more.

plot(x,y,type="n") # What's going on here?
points(x[y>0], y[y>0], col="red")
points(x[y<0], y[y<0], col="blue")

plot(x,y,type="n")
lines(x,y) # Any difference from plot(x,y,type"l")
```

```
plot(x,y,type="l", lty=3)
plot(x,y,type="l", lty=2, lwd=3)
abline(h=0, lty=3, col="green") # horizontal line
abline(v=15, lty=3, col="red") # vertical line


# N.B.
# 1. type - what type of plot to be drawn.
     For example: type="h".
# 2. col - colour of the points.
     For example: col="purple" or col=5.
# 3. lty - line type
     For example: lty="dashed" or lty=3.
# 4. lwd - line width
     For example: lwd=2.
```

Module 22: Plotting - boxplot

Boxplot is a useful tool to investigate the data by presenting summary quantities in the plot. Run the following R codes and understand what each line does.

Example:

```
a <- 1:20
boxplot(a)

b <- matrix(sample(1:50),10)
b
boxplot(b)
boxplot(b, col=2:6) # What does col=2:6 do?


sales <- data.frame(Jan=rnorm(10), Feb=rnorm(10),
                    Mar=rnorm(10), Apr=rnorm(10))
sales
boxplot(sales)
```

Module 23: Plotting - text and annotation

You can add text, and other annotation to a plot that you have drawn.

Example:

```
x <- 11:20
y <- rnorm(10)
z <- rnorm(10, sd=0.5) # Note that you are changing the
                       # standard deviation only

plot(x,y, pch=1:10) # pch="points character"
                    # pch=1:10 gives all 10 points
                    # a different character each

plot(x,y, pch="y")
points(x,z, pch="z")
lines(x,y, lty=3, col="red")
lines(x,z, lty=3, col="blue")

plot(x,y)
text(x,y+0.1, LETTERS[1:10]) # What does +0.1 do?
# Two points have the text outside margin.
# Solution: set the limit for y axis

plot(x,y, ylim=c(min(y)-0.1, max(y)+0.1))
text(x,y+0.1, LETTERS[1:10])

# Can you put the letters on the left hand side of the points?

set.seed(100)
year <- 2001:2010
sales <- sort(rexp(10, 0.01)+20) # What are the arguments
                                 # in the rexp command?
number.of.branch <- c(10,10,11,12,13,13,14,17,20,25)
sales.per.branch <- round(sales/number.of.branch,1)
# What are the arguments in the round command?

plot(year,sales, ylim=c(min(sales)-15, max(sales)+15),
xlab="Year", ylab="Sales (Million Pounds)",
main="Fake Company Ltd.")
```

```
text(year,sales+10, number.of.branch)
text(year,sales-10, sales.per.branch)
lines(year,sales,lty=3)

#.................
set.seed(100)
year <- 2001:2010
sales <- sort(round(runif(10,5,40),1))
sales # What does runif do?

sales <- cbind(sales,sort(round(runif(10,5,60),1)))
sales # cbind takes a sequence of vector, matrix or
      # data-frame argument and combine by column

sales <- cbind(sales,sort(round(runif(10,5,80),1)))
sales <- data.frame(sales)
names(sales) <- paste("Branch",LETTERS[1:3],sep=" ")
             # What does ‘paste’ do?
sales # See the structure of the data

# There are several ways we can present the information
# in ‘sales’ into figures.

par(mfrow=c(2,2)) # Creates 2 by 2 panel for figures

# First alternative
plot(year, sales[,1], pch=1, xlab="Year",
ylab="Sales (millions)", main="First alt.",
ylim=c(min(sales)-10, max(sales)+10))
points(year, sales[,2], pch=2)
points(year, sales[,3], pch=3)
legend(2001,80, names(sales), pch=1:3)

# Second alternative
plot(year, sales[,1], type="l", xlab="Year",
ylab="Sales (millions)", main="Second alt.",
ylim=c(min(sales)-10, max(sales)+10))
lines(year, sales[,2], lty=2)
lines(year, sales[,3], lty=3)
legend(2001,80, names(sales), lty=1:3)
```

```
# Third alternative
plot(year, sales[,1], pch=19, xlab="Year",
ylab="Sales (millions)", main="Third alt.",
ylim=c(min(sales)-10, max(sales)+10))
points(year, sales[,2], pch=19, col=2)
points(year, sales[,3], pch=19, col=4)
legend(2001,80, names(sales), pch=19, col=c(1,2,4))

# Fourth alternative
plot(year, sales[,1], type="l", xlab="Year",
ylab="Sales (millions)", main="Fourth alt.",
ylim=c(min(sales)-10, max(sales)+10))
lines(year, sales[,2], col=2)
lines(year, sales[,3], col=4)
legend(2001,80, names(sales), lty=1, col=c(1,2,4))
```

Module 24: Exercises

Answer the following questions regarding plots.

1. (a) Go to the MATH2735 module website:
   http://www1.maths.leeds.ac.uk/∼arief/MATH2735/
   and go to the heading “Dataset”.
   (b) Read the dataset called “pullover” to your R session using the command `read.table()`. The dataset contains the information on *sales* (number of sold pullovers), *price* (price per item in Euro), *advertisement* (cost in local newpaper in Euro), and *hours* (presence of a sales assistant in hours per period).
   (c) Create some plots that would show the potential relationships between *sales* and *price*, *advertisement*, and *hours* (individual pair).

2. (a) Read the table located in:
   http://www1.maths.leeds.ac.uk/∼arief/R/smallcafe.txt into your R session. The table contains sales data of a small cafe in Leeds from 2001 to 2010. The column *Sales* contain the turnover (in thousands of poundsterling), *Cost* contains the operating cost (in thousands of pound-sterling), and *Drinks* contain the percentage of sales come from drinks (both hot and cold drinks).
   (b) In the data frame, create another column for gross profit (in thousands of pound-sterling). This is calculated as the difference between the sales and cost.
   (c) Now plot the sales across the year using solid black circle points.
   (d) In the same plot, draw another set of solid red rectangular points for cost, and solid green triangle points for gross profit.
   (e) Connect the points for each column using dotted lines with their respective colour.
   (f) Add the information of the percentage of drinks sales from the overall sales as texts above each sale point.

3. (a) Go to the MATH2735 module website:
   http://www1.maths.leeds.ac.uk/∼arief/MATH2735/
   and go to the heading ”Dataset”.
   (b) Read the dataset called “thiamin” to your R session using the command `read.table()`. The dataset contains the thiamin (Vitamin B1) content (mg/kg) in cereal grains (wheat, barley, maize, and oats). Create some boxplots of the thiamin content across different cereal grains (in a single panel).

Module 25: Iteration - `for()`

Run the following R commands and understand what each line does.

Example:

```
for(j in 1:10) print(j)

# Run this section together ----
for(j in 1:10)
print(j)
print(j+100)
# End of running a section --
# Why do we not get 101 to 109?

for(j in 1:10){
print(j)
print(j+100)
}
# Do we get 101 to 110 now?
# What has changed?

counter <- 1:10
for(j in counter){
print(j)
print(LETTERS[j])
} # What happens here?

mycities <- c("London", "Manchester", "Edinburgh", "Leeds")
for(j in mycities){
print(j)
}

mylist <- list(a=1:10, b=matrix(1:4,2), d=LETTERS[1:5])
length(mylist)
for(j in mylist){
print(summary(j))
}

# Nested loop
for(j in 1:5){
```

```
   for(k in LETTERS[1:4]){
     print(j)
     print(k)
   }
}

# Which loop structure is executed in entirety first?
# Is it the outside or the inside loop structure?
```

Module 26: Exercises

Answer the following questions and write the correct R command.

1. Prime numbers are just beautiful. They are the building blocks and bricks of the universe. Now, using `for()` loop that evaluate integers from 1 to 100, print only the prime numbers between 1 and 100.

2. In statistics, variance-covariance matrix is commonly involved in many analysis. Create a **data** matrix of size 20×10 first by performing the following command:

   ```
   set.seed(1234) # seed for random number generation
   X <- matrix(rnorm(200),20)
   ```

   Although there is a quick way to construct the variance-covariance matrix using `var()`, do not use that function. Instead, use the `for` loop to fill each of the elements of the 10×10 variance-covariance matrix, row-by-row or column-by-column, whichever is more convenient for you.

   Hint: Consider a data matrix $\mathbf{X} \equiv \{\mathbf{x}_1\mathbf{x}_2 \cdots \mathbf{x}_p\}$ where $\mathbf{x}_i$, $i = 1, \ldots, p$, is vector of size n, i.e. $\mathbf{x}_1 \equiv (x_{11}x_{12} \ldots x_{1n})^T$. The (sample) variance-covariance matrix of $\mathbf{X}$, denoted by $\mathbf{S}$, has the elements

   $$s_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} (x_{ik} - \overline{\mathbf{x}}_i)(x_{jk} - \overline{\mathbf{x}}_j)$$

   for $i = 1, 2, \ldots, p$ and $j = 1, 2, \ldots, p$, and $\overline{\mathbf{x}}_i = \frac{1}{n}\sum_{k=1}^{n} x_{ik}$.

3. The above variance-covariance matrix $\mathbf{S}$ is a symmetric matrix. If you have not done so in the question above, construct $\mathbf{S}$ using **`for()`** loop by filling in the upper diagonal part (including the diagonal), and "mirror" them for the lower diagonal. So, there should be two different uses of the `for()` loop. The first one is to create the upper diagonal matrix, and the second one is to "mirror" the upper diagonal matrix into a full matrix.

Module 27: Iteration - `for()`

We can create an iteration using `while()`. The function will keep iterating the subsequent `R` commands if the condition inside the brackets is true. Run the following `R` codes and understand what each line does.

Example:

```
m=1  # declare m
while(m<10){
print(m)
m=m+1
} # End of while
  # What values are printed?
  # What is the FINAL value of m?
```

Module 28: Exercises

Answer the following questions using the `while()` function:

1. Simulate a moving average process with the standard normal random variable as the driver of the process and $x_0 = 100$. Simulate this process with `while()` function and stop the process when x_i is less than 50 or more than 150 or i has reached 1000. So, discarding x_0, the length of the simulated vector should be at most 1000. In each iteration, there should be only *one* value to be randomly generated from the standard normal distribution.

2. Hypergeometric distribution is a discrete probability distribution to model the number of 'fails' until one 'success'. Using the `while()` function, generate *one* random number from a hypergeometric distribution where in each iteration you flip an unfair coin with 80% probability to give 'fail'. The number that you need to flip the coin should follow a hypergeometric distribution.

3. Now, instead of generating *one* random hypergeometric variable, generate 100 random numbers using the `while()`. So, there will be two nested `while()` loop. The outer loop will generate the 100 random numbers, while the inner loop will generate *one* random number by flipping the coin a number of times.

Module 29: Functions

When we have some group of commands that we use, or for simplicity, we can create a function that will execute a group of R commands. The syntax is:

Example:

```
name.function <- function("list of arguments" ){
.....
.....
return("an R object to be returned")
} # End of function
```

Run the following R commands and understand what is going on in each function.

```
mymean <- function(x){
  mm <- sum(x[is.finite(x)])/length(x[is.finite(x)])
  return(mm)
}

x1 <- c(1:10); x1
x2 <- c(1:9, NA); x2
mean(x1)
mymean(x1)
mean(x2)
mymean(x2)
#-----------------
```

```
mysummary <- function(x){
x <- as.matrix(x)
meanx <- NULL
medianx <- NULL
  for(j in 1:ncol(x)){
    column.mean <- mean(x[,j], na.rm=T)
    column.median <- median(x[,j], na.rm=T)
    cat("Column ",j," Mean=",column.mean,
    " Median=",column.median,"\n")
    meanx <- c(meanx, column.mean)
    medianx <- c(medianx, column.median)
  }
result <- list(mean=meanx, median=medianx)
return(result)
}

z <- matrix(1:30,10)
z
mysummary(z)
mysummary(1:20)


# What does the line ‘meanx <- NULL’ do?
# Recall from Module 19 what does the command ‘cat()’ do?
```

Module 30: Exercises

Answer the following questions regarding the creation of function in R .

1. Create a function that would take a single integer number as input, and will *print out* whether the number is even, odd, or zero.

2. Create a function that would take a single *positive* integer number which is greater than 4 as input, and will *return* a vector of prime numbers from 1 to the number.

3. Create the following data frame with two columns. The first column, named "Year", has the entries of 2001 to 2006. The second column, named "Profit", has the entries 4.5, 2.1, 3.9, 3.6, 2.5, 4.2. Now, create a function that would take the data frame as an input, and *plot* the profit across the years. The horizontal axis should be labelled "Year", and vertical axis should be "Profit (millions)". The numbers should be plotted with points represented in solid rectangle, which are coloured red if they are less than 3 and blue if they are more than or equal to 3. The points should be connected with dotted black line.

4. Create a function that would take a matrix as input, and calculate a one-sample t-statistic for each row in the matrix. The output of the function is the t-statistics from the rows of the matrix.

5. Create a function as an extension of the above function. Instead of returning the t-statistics only, create a new function that would return a list, which contains the t-statistics, mean, standard deviation, and p-value of the (one-sample) t-test, from the rows of the matrix.

Module 31: Functions - *tagged* arguments, or *positional* arguments

In R, functions can have *tagged* arguments, or *positional* arguments. To find out the names of the tags, you have to look at the help pages. The arguments are matched as follows (in order):

1. exact matching on tags
2. partial matching on tags
3. positional matching

Consider the following R codes and understand what is going on in each line.

Example:

```
x <- 1:10
y <- rnorm(10)
par(mfrow=c(2,2)) # 2 by 2 panel
plot(x,y) # What is going on here?
plot(y=x,x=y) # What is going on here?
#---------------------------

x <- c(1:10); mean(x)
y <- c(1,1,2,2,3,3); mean(y)

# Check the sign of t-statistic
# and group means
t.test(x,y)
t.test(x=x, y=y)
t.test(y=y, x=x)
t.test(x=y, y=x)
t.test(y=x, x=y)
t.test(1:10, c(1,1,2,2,3,3))
```

Module 32 - Display the structure of an R object

Some R function either do not return any value, return a single value/number, or return something more complex than that. Once you store the results of the function into an R object, you can understand the structure of information in that R object by using the command `str()`. Once you have understood the structure, you can extract relevant information from the object. Run the following R codes and understand what is going on.

Example:

```
result1 <- t.test(1:5, 1:10)
str(result1) # What do you see?
result1$statistic # t-statistic
result1$p.value # p.value of testing H0 that
                # the two sample have the same mean
```

Module 33: Exercises

Answer the following questions:

1. Load an RData from the following location:
   http://www1.maths.leeds.ac.uk/∼arief/matrix-x.RData into your R session.
2. For *each* row of `x`, calculate a two-sample t-statistic from the first five columns and the last five columns using `t.test()`. Create a vector to store the t-statistics only, another vector to store the p-values of the tests, and another vector to store the difference in group means. The creation of the vectors must be done in a `for()` loop.
3. Create a data frame from the vectors above: t-statistics, p-values, and mean differences.