MAS1802: Problem Solving II (Statistical Computing with R)

Dr Lee Fawcett

School of Mathematics, Statistics & Physics

Semester 2, 2017/2018

1.1 Housekeeping

MAS1802 – A tale of two halves:

- Weeks 1–5: Pencil-and-paper problem solving with Evgenios Kakariadis
- Weeks 6–12: Statistical computing with Lee Fawcett
- Week 12: Revision for the exam with Evgenios

These slides refer to the computing part only.

1.1 Housekeeping

Computing – General course structure:

Type of session	Day	Time	Place
Lecture	Wednesday	11–12	Herschel LT1
Office hour	Thursday	12–1	My office (2.07)
Practical	Friday	11–1	Herschel PC / Barbara Strang Side PC
Drop-in	Monday	12–1	Herschel PC
Deadline:	*Friday*	*11.00am*	*General office/online*

- This week: No Monday drop-in, extra lecture, extra practical!
- Next week: Monday drop-in $\longrightarrow$work due in on Friday (deadlines not set properly in NESS yet)

1.1.1 Lecturer information: who am I?

- The module leader, and lecturer for the computing part of this module, is Dr. Lee Fawcett
- If you have any questions, comments or feedback on this course I can be contacted via email
- More complicated questions about the course are best dealt with face-to-face

1.1.2 Lectures

- Fewer lectures for this course than for other modules (nature of the course)
- The calendar at the front of this handout contains a full schedule for the computing part of this course
- A summarised version of the notes will be used as slides - interspersed with quizes and other really fun things. Pay attention!
- Notes, slides and practical sheets will be posted to BlackBoard, and a dedicated webpage, soon after the lecture/practical session
- Lectures will be recorded via ReCap
- Check your `ncl` e-mail regularly: class cancellations etc.

1.1.3 Computer practicals

- Lots of support - make good use of this!

- Attendance closely monitored

1.1.3 Computer practicals

- You are more likely to get high marks for the coursework if:
  - You have attended the lecture: In practicals, I will assume you have been to the lecture on Monday; this is reflected in what is expected of you in the practicals
  - You don't panic! Read the notes carefully, and type code **slowly**
  - You ask for help
  - You try to have fun
  - You are organised!

1.1.4 Office hours

- Thursday 12-1pm
- Any other office hours (e.g. MAS8306)
- Or just give me a knock (usually not free Wednesday afternoon of Friday after 2pm)

1.1.5 Assessment

- End of semester examination worth **40%** of your overall mark. This will include material from the pencil-and-paper part of the course after Easter
- Assessment of course-work worth **60%** of your overall mark:
  - 'Weekly' computing exercises (10%: 3@ 3.33% each)
  - Group computing project (30%)
  - Problem-solving exercises (20%)

The weekly computer exercises will be due on the following dates, all of which are Fridays at 11am:

16th March; 20th April; 18th May.

The group computing project will be due in by no later than 11am, Friday 18th May.

1.1.6 Quiz questions

- Last summer, we wrote lots of quiz questions in `Numbas` to help develop your computing skills for this part of the module
- These quiz questions are written within the usual computer-based assessment (CBA) framework – if you're interested, go to:

  `www.mas.ncl.ac.uk/~nlf8/basicR`

- I will make lectures more fun by including quiz questions here, too!

1.2 What is **R**?

- **R** is a computer package that is widely used
- For example Google, IBM, Shell and spread betting companies use **R** for statistical software development and data analysis
- See the NY times article at http://tinyurl.com/9m5qyh.
- **R** uses a command line interface, though several graphical user interfaces are available
- Why use a command line interface as opposed to Excel, say?

1.2 What is **R**?

- **R** is highly extensible through the use of user-submitted libraries for specific functions or specific areas of study
- A particular strength of **R** is it's graphical facilities, which produce quality graphs (better than MatLab!) that can include mathematical symbols
- **R**'s syntax is similar to that used in MatLab
- **R** has loads of built-in functions for statistical analysis, but can also do lots of basic mathematical operations too
- **R** is completely free (unlike MatLab), as is the Rstudio software which we will use in this course
- Named after Ross Ihaka and Robert Gentleman (Auckland University, NZ) who developed the software
- Industry-standard statistical software, and used right throughout your degree

1.2.1 Accessing **R**

- R is installed on all University machines
- In this class we will be using Rstudio, which is an R IDE
- Both R and Rstudio are free, so you can install them on you own computer. See:

  `http://www.ncl.ac.uk/maths-physics/students/resources/r`

1.2.2 Previous computing knowledge

- This course is intended to teach you the basics of programming in **R**
- **No** previous programming knowledge is assumed, although MatLab should have helped your general computing skills
- It is crucial that you come to all the practicals for this course
- It is also vital that you let me/other staff know, in the practical sessions, if you're not sure about anything... *Burying your head in the sand will not help. Neither will crying, or getting angry...*

1.2.3 Recommended R textbooks

- At `http://goo.gl/nzQTK`, I have constructed a list of suitable books (with comments) on R programming
- Since you will be using R throughout your degree, then it may be worthwhile buying a good R textbook...
- ... however, a really good, free-to-download book, written by Venables and Smith, can be found here:

  `https://cran.r-project.org/doc/manuals/R-intro.pdf`

1.3 Movies dataset

- The internet movie database (IMDB) is a website devoted to collecting movie data supplied by studios and fans.
- It claims to be the biggest movie database on the web and is run by amazon. More information about imdb.com can be found online.
- Movies were selected for inclusion if they:
  - had a known length;
  - had been rated by at least one IMDB user;
  - had an MPAA rating.

1.3 Movie dataset

- **Title**. Title of the movie
- **Year**. Year of release
- **Budget**. Total budget in US dollars. If the budget isn't known, then it is stored as '-1'
- **Length**. Length in minutes
- **Rating**. Average IMDB user rating
- **Votes**. Number of IMDB users who rated this movie
- **r1-10**. Percentage (to nearest 5%) of users who rated this movie a 1,..., 10

1.3 Movies dataset

- **mpaa**. MPAA rating
- **Action**, **Animation**,.... Binary variables representing if movie was classified as belonging to that genre. A movie can belong to more one genre. See for example the film *Ablaze* in Table 1.1
- There are a total of 24 variables and 4988 films
- This is **only a subset** of the data, the actual data set contains information on over 50,000 movies

				Voting statistics					Movie genre							
Title	Year	Length	Budget	Rating	Votes	r1	...	r10	mpaa	Action	Animation	Comedy	Drama	Documentary	Romance	Short
A.k.a. Cassius	1970	85	-1	5.7	43	4.5	...	14.5	PG	0	0	0	0	1	0	0
AKA	2002	123	-1	6.0	335	24.5	...	1.5	R	0	0	0	1	0	0	0
Alien Vs. Pred	2004	102	45000000	5.4	14651	4.5	...	4.5	PG-13	1	0	0	0	0	0	0
Abandon	2002	99	25000000	4.7	2364	4.5	...	4.5	PG-13	0	0	0	1	0	0	0
Abendland	1999	146	-1	5.0	46	14.5	...	24.5	R	0	0	0	0	0	0	0
Aberration	1997	93	-1	4.8	149	14.5	...	4.5	R	0	0	0	0	0	0	0
Abilene	1999	104	-1	4.9	42	0.0	...	24.5	PG	0	0	0	1	0	0	0
Ablaze	2001	97	-1	3.6	98	24.5	...	14.5	R	1	0	0	1	0	0	0
Abominable Dr	1971	94	-1	6.7	1547	4.5	...	14.5	PG-13	0	0	0	0	0	0	0
About Adam	2000	105	-1	6.4	1303	4.5	...	4.5	R	0	0	1	0	0	1	0

1.4 Useful R package

This module has an associated R package for quick access to datasets (e.g. the movies dataset). Installing this package is straightforward:

```
> install.packages("ncldata",
+          repos="http://R-Forge.R-project.org",
+          type="source")
```

To load the package, use:

```
> library(ncldata)
```

Then:

```
> data(movies)
```

Part II

Introduction to R

2.1 A simple R session

At its most basic, R can be used as a calculator – for example, for multiplication and division...

```
> 5*5
[1] 25
> 10.2/6
[1] 1.7
```

...and more ‘advanced’ operations:

```
> 2^3
[1] 8
```

```
> exp(1.5)
[1] 4.481689
```

```
> log(10)
[1] 2.302585
```

```
> 4 %% 3
[1] 1
```

2.1.1 Assignment operations

- In the computer practicals we will use assignment, i.e. `x = 5`.
- However, computer assignment is different from typical mathematical assignment.
- For example,

```
> x = 5
> x = x + 1
> x
[1] 6
```

- Notice that when we type `x = 5`, R doesn't display or print any output to the screen.
- If we want to see what value has been assigned to the variable, we type `x`.
- An alternative is to surround the expression with brackets.
- For example

```
> (x = 2*x)
[1] 12
```

- You can also use the `<-` operator for assignment. This is, for almost all situations, identical to the = operator.

Question 1

What is the value of x?

```
> x = 5
> x = 2*x + 1
> x
```

1. 5
2. 11
3. 12

Question 1

What is the value of x?

```
> x = 5
> x = 2*x + 1
> x
[1] 11
```

1. 5
2. 11
3. 12

Question 2

What is the value of x?

```
> x = 5
> y = 6
> x = x + 2*y
> x + 5
> x
```

1. 5
2. 10
3. 17
4. 22

Question 2

What is the value of x?

```
> x = 5
> y= 6
> x = x + 2*y
> x + 5
[1] 22
> x
[1] 17
```

1. 5
2. 10
3. 17
4. 22

Question 3

What is the value of x?

```
> x = 5
> y = 6
> x + 5
> x
```

1. 5
2. 10
3. 17
4. 22

Question 3

What is the value of x?

```
> x = 5
> y = 6
> x + 5
[1] 10
> x
[1] 5
```

1. 5
2. 10
3. 17
4. 22

2.1.2 Data types

R has a variety of data types:

```
> (v = TRUE)
[1] TRUE
```

```
> (w = "fred")
[1] "fred"
```

```
> (x = 5.0)
[1] 5
```

And some “special" data types:

```
> (y = 5/0)
[1] Inf
> (z = y-y)
[1] NaN
```

Type	Example 1	Example 2	Example 3	Example 4
Doubles	2	3.1242	-45.6	4e-10
Logicals/Boolean	TRUE	FALSE		
Characters	“FRED”	“x”	“Male”	“TRUE”
Infinity	Inf	5/0		

Table : Summary of data types in R.

Another important data type in R is `NA`. This is used to represent missing values. A list of data tables is given in Table 1.1.

Question 4

What type of variable is x?

```
> x = "1"
```

1. Character
2. Double
3. Logical

Question 4

What type of variable is x?

```
> x = "1"
```

1. Character
2. Double
3. Logical

2.2 The R workspace

Once you create a variable, R stores that variable in memory for reuse. You can view available variables with the `ls()` command:

```
> ls()
[1] "movies"
```

To delete a variable in R, we use the `rm` function. For example,

```
> x = 0
> y = 1
> z = 2
> ls()
[1] "movies" "x"      "y"      "z"
```

```
> rm(x)
```

```
> ls()
[1] "movies" "y"      "z"
```

The nuclear option

We can remove everything in the workspace using:

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

2.3 Vectors

- Vectors are the most basic of all data structures, but are used in almost all R code.
- A R vector contains *n* values of the same type, where *n* can be zero.
- For example

```
> c(0, 1, 2, 3, 4, 5)
[1] 0 1 2 3 4 5
> (my_first_vec = c(0, 1, 2, 3, 4, 5))
[1] 0 1 2 3 4 5
> (my_second_vec = c("Male", "Female", "Male"))
[1] "Male"   "Female" "Male"
```

In R, when we type:

```
> x = 5
> y = "Fred"
```

We have actually created a vector of doubles and characters (of length one).

There are special functions in R to determine type of a variable:

```
> x = 5
```

```
> is.double(x)
[1] TRUE
```

```
> is.character(x)
[1] FALSE
```

```
> is.vector(x)
[1] TRUE
```

To determine the length of a vector in R, we use the `length` function:

```
> length(my_first_vec)
[1] 6
> length(my_second_vec)
[1] 3
```

To create sequences of numbers we use the `seq` command. For example,

```
> (x1 = seq(1, 6))
[1] 1 2 3 4 5 6
```

```
> (x2 = seq(-4, 4, by=2))
[1] -4 -2  0  2  4
```

Command description	Example	Result
Length	`length(x)`	4
Reverse order	`rev(x)`	3,5,5,1
Sort	`sort(x)`	1,3,5,5
Sum	`sum(x)`	14
Extract unique elements	`unique(x)`	1,5,3
Indices of particular elements	`which(x==5)`	2,3

Table : Useful vector functions. In the above examples, `x = c(1,5,5,3)`. Check the associated R help for further information.

Question 5

What is the value of x?

```
> x = seq(2, 6)
```

1. 2,3,4,5,6
2. 2,6
3. 6,4,2

Question 5

What is the value of x?

```
> x = seq(2, 6)
> x
[1] 2 3 4 5 6
```

1. 2,3,4,5,6
2. 2,6
3. 6,4,2

Question 6

What is the value of `blob`?

```
> blob = seq(2, -2, by=-2)
```

- -2 -1 0 1 2
- -2 0 2
- 2 0 -2
- 2 1 0 -1 -2

Question 6

What is the value of `blob`?

```
> (blob = seq(2, -2, by=-2))
[1]  2  0 -2
```

- -2 -1 0 1 2
- -2 0 2
- 2 0 -2
- 2 1 0 -1 -2

2.3.1 Vector operations

- When our data is in a vector structure, we can apply standard operations to the entire vector.
- For example,

```
> (x = seq(-4, 4))
[1] -4 -3 -2 -1  0  1  2  3  4
```

```
> x*x
[1] 16  9  4  1  0  1  4  9 16
```

```
> x - 5
[1] -9 -8 -7 -6 -5 -4 -3 -2 -1
```

```
> x + x
[1] -8 -6 -4 -2  0  2  4  6  8
```

Question 7

What is the value of z?

```
> x = c(1,10,100,1000)
> y = c(1, 5, 10, 100)
> z = x/y
```

1. 1, 10, 100, 1000
2. 1, 5, 10, 1000
3. 1, 1, 1, 1
4. 1, 2, 10, 10

Question 7

What is the value of z?

```
> x = c(1,10,100,1000)
> y = c(1, 5, 10, 100)
> (z = x/y)
[1]  1  2 10 10
```

1. 1, 10, 100, 1000
2. 1, 5, 10, 1000
3. 1, 1, 1, 1
4. 1, 2, 10, 10

Question 8

What is the value of y?

```
> x = seq(1, 5)
> y = length(x + 1)
```

1. 1
2. 5
3. 10
4. 12

Question 8

What is the value of y?

```
> x = seq(1, 5)
> (y = length(x + 1))
[1] 5
```

1. 1
2. 5
3. 10
4. 12

2.3.2 Extracting elements from vectors

- R has a number of useful of methods that we use to extract subsets of our data.
- For example to pick out particular elements:

```
> my_first_vec[2]
[1] 1
> my_second_vec[2:3]
[1] "Female" "Male"
> my_first_vec[4:2]
[1] 3 2 1
```

- We can also use other arguments.
- For example to remove the last entry in the vector, we use the `length` function:

```
> l = length(my_first_vec)
> my_first_vec[1:(l-1)]
[1] 0 1 2 3 4
```

- We determine the length of the vector using the `length` function and select particular elements using the `[·]` operator.

Question 9

What is the value of x?

```
> y = seq(2, 6)
> x = y[3:length(y)]
```

1. 2,3,4,5,6
2. 3,4,5,6
3. 4,5,6
4. 3

Question 9

What is the value of x?

```
> y = seq(2, 6)
> (x = y[3:length(y)])
[1] 4 5 6
```

1. 2,3,4,5,6
2. 3,4,5,6
3. 4,5,6
4. 3

2.4 Logical vectors

Boolean	A	B	$\bar{A}$	$\bar{B}$	$A \cap B$	$A \cup B$
R	A	B	!A	!B	A & B	A | B
	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE

Table : Truth table for Boolean operation

Question 10

What is the value of z?

```
> x = TRUE
> y = FALSE
> z = x & !y
```

1. TRUE
2. FALSE

Question 10

What is the value of z?

```
> x = TRUE
> y = FALSE
> (z = x & !y)
[1] TRUE
```

1. TRUE
2. FALSE

Question 11

What is the value of z?

```
> x = TRUE
> y = FALSE
> z = 5
> x = (y | !y)  & x
> (z = x)
[1] TRUE
```

1. TRUE
2. FALSE

Question 11

What is the value of z?

```
> x = TRUE
> y = FALSE
> z = 5
> x = (y | !y)  & x
> z = x
```

1. TRUE
2. FALSE

2.4.1 Using logicals for sub-setting vectors

We can construct vectors of logical operators and use them to take subsets of vectors. For example:

```
> (logic1 = c(TRUE, FALSE, TRUE, FALSE))
[1]  TRUE FALSE  TRUE FALSE
> (vec1 = seq(1, 4))
[1] 1 2 3 4
> vec1[logic1]
[1] 1 3
```

2.4.2 Relational operators

When programming it is often necessary to test relations for equality and inequality. To do this in R we use the relation operators. First let's define some variables:

```
> x = 5
> y = 7
```

To test for equality we use ==:

```
> x == 5
[1] TRUE
> x == y
[1] FALSE
```

Similarly, to test for inequality we use `!=`:

```
> x != 5
[1] FALSE
> y != x
[1] TRUE
```

There are also commands for greater/less than:

```
> y > 6
[1] TRUE
> x >= 5
[1] TRUE
> x <= y
[1] TRUE
```

We can also apply these techniques to vectors. For example:

```
> (vec2 = seq(0, 10, by=2.5))
[1]  0.0  2.5  5.0  7.5 10.0
> vec2 > 3
[1] FALSE FALSE  TRUE  TRUE  TRUE
> vec2 < 9
[1]  TRUE  TRUE  TRUE  TRUE FALSE
> (vec2 > 3) & (vec2 < 9)
[1] FALSE FALSE  TRUE  TRUE FALSE
```

2.4.2 Relational operators

Operator	Tests for	Example	Result
$==$	Equality	`x == 5`	`TRUE`
$!=$	Inequality	`x != 5`	`FALSE`
$<$	Less than	`x < 5`	`FALSE`
$<=$	Less or equal	`x <= 5`	`TRUE`
$>$	Greater	`x > 5`	`FALSE`
$>=$	Greater or equal	`x >= 5`	`TRUE`

Table : Summary of R relational operators. The example is for `x = 5`.

Using relational operators allows us to extract subsets of data very easily. Consider the movie budgets:

```
> length(Budget)
[1] 4988
```

To select movies where the budget is known, we use the following command...

```
> non_zero_b = Budget[!is.na(Budget)]
> length(non_zero_b)
[1] 3990
```

... and to select movies where the movie length is greater than 60 mins but shorter than 90 mins

```
> m_l = Length[Length > 60 & Length < 90]
```

Question 12

What is the value of z?

```
> x = seq(-2, 10)
> z = x[(x >= 7) ]
```

- TRUE
- 7,8,9,10
- -2, -1, 7,8,9,10
- 0,1,2,3,4,5,6

Question 12

What is the value of z?

```
> x = seq(-2, 10)
> (z = x[(x >= 7) ])
[1]  7  8  9 10
```

- TRUE
- 7,8,9,10
- -2, -1, 7,8,9,10
- 0,1,2,3,4,5,6

Question 13

What is the value of z?

```
> x = seq(-2, 10)
> z = x[(x >= 7) | x <0]
```

- TRUE
- 7,8,9,10
- -2, -1, 7,8,9,10
- 0,1,2,3,4,5,6

Question 13

What is the value of z?

```
> x = seq(-2, 10)
> (z = x[(x >= 7) | x <0])
[1] -2 -1  7  8  9 10
```

- TRUE
- 7,8,9,10
- -2, -1, 7,8,9,10
- 0,1,2,3,4,5,6

2.5 Data frames

- A data frame is a special kind of object.
- We use data frames for storing and managing data sets that have a rectangular structure.
- Typically the rows correspond to *cases* and the columns to *variables*.
- The crucial difference between a data frame and a `matrix` is that all values in a matrix must be of the same type.
- The next code segment constructs a simple data frame.

2.5 Data frames

First, we construct three vectors:

```
> age = c(24, 26, 25, 21)
> sex = c("Male", "Female", "Male", "Female")
> respond = c(TRUE, FALSE, FALSE, FALSE)
```

Then we combine them using the `data.frame` function:

```
> (df1 = data.frame(age=age, gender=sex,
+    respond=respond))
  age gender respond
1  24   Male    TRUE
2  26 Female   FALSE
3  25   Male   FALSE
4  21 Female   FALSE
```

2.5 Data frames

- The data frame `df1` has three columns and four rows.
- Once we put our data into a data frame, then data manipulation is easier.
- To calculate the dimensions of a data frame we use `dim`:

```
> dim(df1)
[1] 4 3
```

2.5 Data frames

To extract the first column we use square brackets:

```
> df1[ ,1]
[1] 24 26 25 21
```

Similarly, we can get the first row

```
> df1[1, ]
  age gender respond
1  24   Male    TRUE
```

2.5 Data frames

The column names are also easily manipulated

```
> colnames(df1)
[1] "age"     "gender"  "respond"
```

```
> (colnames(df1) = c("Age", "Sex", "Respond"))
[1] "Age"     "Sex"     "Respond"
```

When we download the `movies` data set, we automatically create a data frame:

```
> dim(movies)
[1] 4988   23
> colnames(movies)[1:4]
[1] "Title"  "Year"   "Rating" "Votes"
> movies[3363, 1:4]
                                                Title Year
Rating  Votes
3363 Star Wars: Episode I - The Phantom Menace 1999
6.5 607932
```

2.5.1 Subsets of data frames

We can also retrieve subsets from the data frame. For example, if we wanted only female responses, then...

```
> (female_only = df1$Sex=="Female")
[1] FALSE  TRUE FALSE  TRUE
> (df2 = df1[female_only, ])
  Age    Sex Respond
2  26 Female   FALSE
4  21 Female   FALSE
```

2.5.1 Subsets of data frames

... or people 25 and over:

```
> over_25 = df1$Age>=25
> (df3 = df1[over_25, ])
  Age    Sex Respond
2  26 Female   FALSE
3  25   Male   FALSE
```

2.5.2 Example: movie data

We can select movies where the budget is greater than $100,000

```
> m1 = movies[movies$Budget > 100000,]
> dim(m1)
[1] 4881   23
```

Or movies that cost more than $100,000 but are not R rated:

```
> m2 = movies[movies$Budget > 100000 & movies$mpaa !=
"R",]
> dim(m2)
[1] 3174  23
```

Or movies that are either PG or PG-13:

```
> m3 = movies[
+   movies$mpaa == "PG" | movies$mpaa == "PG-13",]
> dim(m3)
[1] 3229   23
```

Question 14

What type of variable is `m1`?

```
> m1 = movies[movies$Budget > 10000,]
```

- A vector of doubles
- A vector of String
- A vector of Logical
- A Data frame

Question 14

What type of variable is `m1`?

```
> m1 = movies[movies$Budget > 10000,]
> is.data.frame(m1)
[1] TRUE
```

- A vector of doubles
- A vector of String
- A vector of Logicals
- A Data frame

Question 15

What type of variable is `m2`?

```
> m2 = movies$Budget > 10000
```

- A vector of doubles
- A vector of Strings
- A vector of Logicals
- A Data frame

Question 15

What type of variable is m2?

```
> m2 = movies$Budget > 10000
> is.vector(m2)
[1] TRUE
> is.logical(m2)
[1] TRUE
```

- A vector of doubles
- A vector of String
- A vector of Logicals
- A Data frame