MAS2906

Computational Probability and Statistics with R

Dr. Lee Fawcett

Semester 1, 2022/23

1 General Housekeeping

1.1 Lecturer information

The module leader for MAS2906 is Dr. Lee Fawcett. If you have any questions about the module he can be contacted via email at `lee.fawcett@newcastle.ac.uk`. Lee is based in room 2.07 on the second floor of the Herschel Building. There will be several other people on the MAS2906 team assisting with this module, primarily in the computer practical sessions. They're all really friendly PhD students – give them a shout in class if you're stuck with anything!

1.2 Classes

As you know, things have been very different at University since March 2020! You will have a blended learning experience this year, meaning you will have some *Present-in-Person* activities on campus and some online activities. For this module we will use a mix of *synchronous* and *non-synchronous* online activities.

- Present-in-Person (PiP): A series of 1 hour, weekly lectures (with Lee), followed by a 2 hour computer practical session (with the full MAS2906 team)
- Non-synchronous online (NSO): Short online activities posted to Canvas every week and to be completed in your own time
- Synchronous online (SO): A series of optional, 1 hour office hours – a chance for you to drop in and ask any questions you might have. No new material will be presented

Full details can be found in the Module Schedule on Canvas.

Students can take part in other NSO activities, should they wish, including interacting with the Canvas discussion board and one-to-one Zoom meetings (by request). And of course, you will all be expected to prepare and complete homeworks in your own time and revise for tests!

1.3 Assessment

MAS2906 has no written exam; you will be assessed via two Numbas tests and a written project:

- Test 1 (**25%**): Covers basic R work from the first few weeks of the module. The test will entirely be completed within the Numbas application. A sample test will be provided, and dedicated time made for revision in class. The test will be open for a one week period, with a deadline of **4pm, Friday 4 November 2022**
- Project (**25%**): The project will cover some of the more theoretical aspects of the course, but will also require you to use some of your R skills from the first few weeks of the module. Students can submit their work in pairs if they wish. The project material will be released three weeks before the deadline of **4pm, Friday 9 December 2021**. Submission will be via Canvas upload

- Test 2 (**50%**): The final test will cover a range of material from across the module. The test will be completed within the Numbas application, but will also require a Canvas upload element. Sample test questions will be provided, and dedicated time will be made for revision in class. The test will take place during the **January assessment period** – precise details will be released later in the semester.

1.4 Late work policy

In this module, deadline extensions can be requested for the project (by means of submitting a PEC form), and work submitted within 7 days of the deadline without good reason will be marked for reduced credit. This module also contains tests worth more than 10% for which rescheduling can be requested (by means of submitting a PEC form). For details of the policy (including procedures in the event of illness etc.) please consult the Mathematics, Statistics & Physics Community pages on Canvas, under: Assessment Information, Late Work and Missed Assessments.

1.5 Course materials

All course materials will be posted to the dedicated MAS2906 Canvas page – where you will find:

- A full copy of these lecture notes, with gaps
- A copy of the lecture notes with the gaps filled in, after the associated PiP/SO/NSO activities
- All slides used to support PiP/SO/NSO activities after these have taken place
- All video recordings
- Solutions to assessed exercises
- Any other materials, including additional handouts/worksheets, links to useful Numbas material, specimen exam papers and solutions, . . .

1.6 Books and other resources

At various points in the module you will be directed to additional reading, should you want extra examples or just another source of information relating to the material presented in the notes. This will primarily be text books (many of which are available to view online), but perhaps other sources (e.g. journal publications, magazine articles etc.).

Some useful books include:

- W.N. Venables and D.M. Smith: An Introduction To R. Free to download at:

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

- D. Braun and D.J. Murdoch: A First Course In Statistical Programming With R
- S. Ross: A First Course in Probability
- N.A. Weiss, P.T. Holmes and H. Michael: A Course In Probability

1.7 General background

In MAS2906 we will learn about computational tools for performing probability calculations and statistical analysis. We will use the R programming language. R is very widely used in universities, businesses and other organizations due to its many advantages over other similar software packages. It is available free of charge, it is 'open source' and can be used for a very wide range of statistical and scientific applications. Libraries of R functions and data-structures have been created and are readily available on the web, enabling statisticians to perform a very wide range of tasks but with a minimum of additional programming. Learning to use R is important because:

(i) It is the foremost tool in modern computational statistics

(ii) It is a great way to learn general concepts in programming which can be used with other languages or in other contexts

(iii) It can be used to illustrate ideas in probability and statistics

In MAS1608 last year we focused mainly on point (ii) above; in this module, we focus mainly on points (i) and (iii).

Given the COVID-19 situation recently, many of you might be worried about not having had the chance to learn R properly. Thus, one of the first things we will do in MAS2906 is review some of the key concepts in R from the MAS1608 syllabus. This will roughly occupy the first two weeks of the module.

For the rest of semester 1, we will present a mixture of background theory in probability and statistics, together with elements of the R programming language. Thus, from around the third week of the course, activities will contain both some statistical theory and some R programming concepts. Most of the theory will be covered in the PiP lectures; in the PiP computer practical sessions, and through your non-synchronous online activities, you will put that theory into practice through lots of worked examples.

It is vital that, if possible, you attend your lectures and practical sessions. This is a great opportunity to ask us questions face-to-face, seek help from other students, and of course prepare for the two tests. It is even more important that you follow instructions on Canvas and complete the other activities by the dates suggested – if you are told that some NSO activities should be completed before the next PiP computer practical, then make sure you do exactly that – the work set in the computer practical class will assume you have completed all work to-date!

2 R Refresher

2.1 Preliminaries

R is installed on all University machines. In this module we will be using RStudio, which is an R *Integrated Development Environment*, or IDE. This just makes R much easier to use and navigate, through the four windows that you see when you open RStudio (text editor, console window, file explorer and graphics display).

You might remember from MAS1608 that both R and RStudio are free, so you can install them on your own computer. Versions of R and RStudio are available for Windows, Apple Mac and Linux. See

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

for more details.

We will assume a limited working knowledge of R and RStudio. The first three weeks are designed to recap the basics from MAS1608. Specifically, we will review (i) data types and vectors in R; (ii) dataframe manipulation; (iii) data summaries; (iv) functions and control statements. Everything we do for the remainder of semester 1 will then assume you have a good grasp of these basics, so please work hard to make sure you're up-to-speed with everything as we progress through this Section of the notes! And of course, get in touch if you're not sure about anything – either by email or via the Canvas discussion board.

You might recall the `movies` dataset from MAS1608. This was obtained from the internet movie database (IMDB): `http://imdb.com/`, which 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 can be found online:

`http://imdb.com/help/show_leaf?about`

This includes information about the data collection process. This is a great dataset to use to demonstrate R's dataframe manipulation and querying capabilities – it is large, has many variables, and is interesting. Everyone loves movies! See Table 1 for an extract of the movies dataset.

				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	NA	5.7	43	4.5	...	14.5	PG	0	0	0	0	1	0	0
AKA	2002	123	NA	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	NA	5.0	46	14.5	...	24.5	R	0	0	0	0	0	0	0
Aberration	1997	93	NA	4.8	149	14.5	...	4.5	R	0	0	0	0	0	0	0
Abilene	1999	104	NA	4.9	42	0.0	...	24.5	PG	0	0	0	1	0	0	0
Ablaze	2001	97	NA	3.6	98	24.5	...	14.5	R	1	0	0	1	0	0	0
Abominable Dr	1971	94	NA	6.7	1547	4.5	...	14.5	PG-13	0	0	0	0	0	0	0
About Adam	2000	105	NA	6.4	1303	4.5	...	4.5	R	0	0	1	0	0	1	0

Table 1: A subset of ten rows of the movie data set. **Credit:** This data set was initially constructed by Hadley Wickham at `http://had.co.nz/`.

We have written an add-on R package for the module which will allow easy access to many interesting datasets (including the movies dataset). Installing the package is straightforward, and will make loading datasets used in the notes really easy:

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

To load the package, type:

```
library(ncldata)
```

Then, to specifically load the movies dataset, type:

```
data(movies)
```

Alternatively, some datasets can be downloaded directly from the course Canvas page, and some from web sources; examples of how to do this will be given later in the notes.

2.2 Data types in R

R has a variety of data types. For example:

```
v = TRUE                    # 'Logical' or 'boolean' - TRUE or FALSE
w = "fred"                  # 'Character' or 'string'
x = 5.0                     # 'Double' or 'numerical'
```

There are also some 'special' data types:

```
y = 5/0                     # Returns Inf
z = y-y                     # Returns NaN
```

Another important data type is `NA`, short for 'Not Available' and used for missing values. A summary of the data types in R is given in Table 2 below. Note the way R presents doubles in standard form (Example 4); notice also that anything declared inside inverted commas will be assumed by R to be of Character type.

Type	Example 1	Example 2	Example 3	Example 4
Doubles	2	3.1242	-45.6	4e-10
Logicals/Boolean	TRUE	FALSE		
Characters	"Fred"	"TRUE"	"Male"	"2"
Infinity	Inf	5/0		
Other	NaN	NA		

Table 2: A summary of some data types in R.

2.3 Vectors

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

```
1 c(0, 1, 2, 3, 4, 5)
2  [1] 0 1 2 3 4 5
3 (first_vec = c(0, 1, 2, 3, 4, 5))
4  [1] 0 1 2 3 4 5
5 second_vec = c("Male", "Female", "Male")
6  [1] "Male"   "Female"   "Male"
7 x = 5.2
8 y = "Fred"
```

Some notes on the above code listing:

- In lines 1-2 we see a vector of length $n = 6$, containing the values $\{0, 1, \ldots, 5\}$. Note that this vector has not been assigned for future use
- In lines 3-4, we see the same vector as in lines 1-2, but this vector has now been assigned to the object "first_vec"
- The outer round brackets used in line 3 are not necessary - they are used in these notes and slides to ensure the resulting vector is printed to the notes/screen, for your convenience so you can see what the code has produced
- "first_vec" is a vector of doubles
- "second_vec" is a vector of $n = 3$ characters
- Both x and y are vectors too – each of length $n = 1$ (x is a vector containing $n = 1$ double; y is a vector containing $n = 1$ character)

There are special functions in R that can be used to determine the variable type. For example:

```
is.double(x)
 [1] TRUE
is.character(x)
 [1] FALSE
is.character(y)
 [1] TRUE
is.vector(first_vec)
 [1] TRUE
```

And of course, we can use the `length` command to determine the length of a vector:

```
length(first_vec)
 [1] 6
length(second_vec)
 [1] 3
```

And you can use the seq command to generate sequence vectors very quickly:

```
(x1 = seq(1, 6))
 [1] 1 2 3 4 5 6
(x2 = seq(-4, 4, by=2))
 [1] -4 -2 0 2 4
(x3 = seq(3, -2, by=-0.5))
 [1]  3.0  2.5  2.0  1.5  1.0  0.5  0.0 -0.5 -1.0 -1.5 -2.0
```

The following code listing demonstrates some other built-in R commands that can be very useful when applied to a vector – `rev`, `sort`, `sum`, `unique` and `which`. Make sure you understand what each of these is doing – especially `unique` and `which`!

```
rev(x1)                 # Reverses the order of the vector
 [1] 6 5 4 3 2 1
sort(x3)                # Sorts the vector into ascending order
 [1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0
sort(second_vec)        # Can be applied to characters for alphabetical order
[1] "Female" "Male"   "Male"
sum(x2); sum(x3); sum(second_vec)
 [1] 0
 [1] 5.5
 Error in sum(second_vec) : invalid 'type' (character) of argument
unique(second_vec)      # Lists all unique elements of the vector
 [1] "Male"   "Female"
which(x2==2)            # Gives index/indices of particular elements
 [1] 4
which(x3<=1)
 [1]  5  6  7  8  9 10 11
```

2.3.1 Vector operations

When our data are 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 2 1 0 1 2 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
```

2.3.2 Extracting elements from vectors

R has a number of useful methods that we can use to extract subsets of our data. For example, we could use square brackets to extract particular elements:

```
first_vec[2]
 [1] 1
second_vec[2:3]
 [1] "Female"  "Male"
first_vec[4:2]
 [1] 3 2 1
(z = 1:10)
 [1]  1  2  3  4  5  6  7  8  9 10
z[z<3 | z>=7]          # We only want values of z that are less than 3 or at least 7
[1]  1  2  7  8  9 10
```

We will come back to extraction from two-dimensional R objects (e.g. dataframes) shortly.

Another useful way of extracting elements from vectors is to use logical vectors. Recall that the logical data type in R can take only the values `TRUE` and `FALSE`. Hence:

```
A = TRUE
B = FALSE
!A        # Read as "Not A"
 [1] FALSE
!B
 [1] TRUE
A & B     # Need "both A and B"
 [1] FALSE
A | B     # Need "either A or B"
 [1] TRUE
```

Then we could declare a vector of such logicals to help us extract from another vector:

```
(logic1 = c(TRUE, FALSE, TRUE, FALSE))
 [1] TRUE  FALSE  TRUE  FALSE
(vec1 = seq(3, 6))
 [1] 3 4 5 6
vec1[logic1]      # Will only extract the elements in vec1 where logic1 is true
 [1] 3 5
```

We can also sum a logical vector. For example:

```
sum(logic1)
 [1] 2
sum(vec1[logic1])
 [1] 8
```

Example 2.1: If you've not already got RStudio installed on the machine you're using, **you must do so now** by following the advice on our student resources webpage:

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

Here you will find some basic resources aimed at students who have never used R before; scroll down, and you'll find information about how to install R on your home PC.

In RStudio, commit the following lines of code. Both lines read in files stored on a webpage. **Do not copy-and-paste this code – you must type it in carefully!**

```
x = scan("http://www.mas.ncl.ac.uk/~nlf8/teaching/mas2906/x.txt")
y = as.logical(scan("http://www.mas.ncl.ac.uk/~nlf8/teaching/mas2906/y.txt",
    what="logical"))
```

Questions about x	Questions about y
(a) What type of data is in the vector x?	(a) What type of data is in the vector y?
(b) How many values are in the vector x?	(b) What is the first value of y?
(c) What is the smallest value in the vector x?	(c) What is the last value of y?
(d) What is the value of x in position 3500?	(d) How many times does TRUE occur?
(e) What is the 97th smallest value in x?	(e) When is the 472nd occurrence of TRUE?
(f) Find $\sum_{i=5612}^{8664} x_i$.	(f) How many occurrences of TRUE are there between positions 504 and 593 (inclusive)?

Your answers about x	Your answers about y
(a)	(a)
(b)	(b)
(c)	(c)
(d)	(d)
(e)	(e)
(f)	(f)

2.4 Dataframes

A dataframe is a special kind of object in R. We use dataframes for storing and managing datasets that have a rectangular structure. Typically the rows correspond to *cases* and the columns *variables*. The crucial difference between a dataframe and a matrix is that all values in a matrix must be of the same type, whereas in a dataframe we can have responses for cases on variables of different types. For example, we might have survey results on 10 individuals, who have each been asked 5 questions. This would give a dataframe with 10 rows and 5 columns (a so-called "10 by 5 dataframe"; generally, we refer to $r \times c$ dataframes). The variable types need not all be the same – the first column could be a vector of doubles recording the individuals' height in metres; the second column could be a vector of logicals taking the value `TRUE` if they like Marmite; the third column could be a vector of characters taking the name of their favourite book, etc.

2.4.1 Creating and querying your own dataframe

You can construct your own dataframe using R's `data.frame` command. For example:

```
height_m = c(1.81, 1.75, 1.58)
weight_kg = c(NA, NA, 70.3)
marmite = c(TRUE, FALSE, FALSE)
book = c("Midnight in Chernobyl", "To Kill a Mockingbird", "Life of Pi")

(small.survey = data.frame(height_m, weight_kg, marmite, book))
  height_m weight_kg marmite                  book
1     1.81        NA    TRUE Midnight in Chernobyl
2     1.75        NA   FALSE To Kill a Mockingbird
3     1.58      70.3   FALSE            Life of Pi
```

So, case 2 in our survey is 1.75 metres tall, does *not* like Marmite and their favourite book is "To Kill a Mockingbird". We do not know the weights of cases 1 and 2 (missing values). Here we have 3 cases and 4 variables, so we have a 3×4 dataframe. If you had a large dataframe and wanted to quickly know its size, you could use the `dim` command (short for "dimension"). For example:

```
dim(small.survey)
 [1] 3  4
```

We can extract elements from a dataframe using square brackets – just as we demonstrated in Section 2.3.2 for extraction from a vector. However, if the columns in our dataframe have labels, we can also use the $ command to access columns directly. For example:

```
small.survey[2, 4]      # Will extract the element from the 2nd row, 4th column
[1] To Kill a Mockingbird

small.survey[ , 4]      # Will extract the elements from all rows in the 4th column
[1] Midnight in Chernobyl    To Kill a Mockingbird    Life of Pi

small.survey$book       # Does exactly the same as line 4!
[1] Midnight in Chernobyl    To Kill a Mockingbird    Life of Pi
```

```
small.survey$book[2]      # Does exactly the same as line 1!
[1] To Kill a Mockingbird

small.survey[3, ]         # Will extract the elements from all columns in the 3rd row
  height_m weight_kg marmite        book
3     1.58      70.3   FALSE Life of Pi
```

Though used less, the `subset` command can be applied to dataframes to extract information from selected columns only. For example:

```
(small.survey2 = subset(small.survey, select = 3:4))
  marmite                  book
1    TRUE Midnight in Chernobyl
2   FALSE To Kill a Mockingbird
3   FALSE             Life of Pi
```

And what if you wanted the height and favourite book of respondents who didn't like Marmite?

```
subset(small.survey,marmite == FALSE,select=c(1,4))
  height_m                  book
2     1.75 To Kill a Mockingbird
3     1.58             Life of Pi
```

2.4.2 The movies dataframe

The Movies dataset is available from within the `ncldata` package. Recall that you can access this in the following way:

```
1 install.packages("ncldata", repos="http://R-Forge.R-Project.org",
2 type="source")
3 library(ncldata)
4 data(movies)
```

- Line 1 installs the `ncldata` package to your machine. You will only need to do this once, and in PiP sessions in the computer cluster this has been done for you already. You will make use of this package for other datasets in Statistics modules, including MAS2903
- Line 3 reads the package into your current working RStudio space – each time you need to access data from within the package you should use this line of code
- Line 4 then pulls the `movies` dataset from the `ncldata` package, ready for you to use

You can now explore the `movies` dataset:

```
dim(movies)       # Looks like there are 4988 movies and 23 variables
 [1] 4988   23

head(movies)      # Dataframe too large to view; let's look at the first six rows
                        Title Year Rating  Votes Length   Budget BudgetCurrency
1                   10,000 BC 2008    5.1 115607    109 1.05e+08            USD
2          101 Dalmatians (I) 1961    7.2 126498     79       NA           <NA>
3         101 Dalmatians (II) 1996    5.7  84527    103 7.50e+07            USD
4              102 Dalmatians 2000    4.9  29328    100 8.50e+07            USD
5         10 Cloverfield Lane 2016    7.2 226726    104 1.50e+07            USD
6 10 Things I Hate About You 1999    7.2 249949     97 3.00e+07            USD

   mpaa Action Adventure Animation Biography Comedy Crime Documentary Drama
1 PG-13   TRUE      TRUE     FALSE     FALSE  FALSE FALSE       FALSE  TRUE
2  <NA>  FALSE      TRUE      TRUE     FALSE   TRUE FALSE       FALSE FALSE
3  <NA>  FALSE      TRUE     FALSE     FALSE   TRUE  TRUE       FALSE FALSE
4  <NA>  FALSE      TRUE     FALSE     FALSE   TRUE FALSE       FALSE FALSE
5 PG-13  FALSE     FALSE     FALSE     FALSE  FALSE FALSE       FALSE  TRUE
6 PG-13  FALSE     FALSE     FALSE     FALSE   TRUE FALSE       FALSE  TRUE

  Fantasy Horror Mystery Romance SciFi Short Thriller
1   FALSE  FALSE   FALSE   FALSE FALSE FALSE    FALSE
2   FALSE  FALSE   FALSE   FALSE FALSE FALSE    FALSE
3   FALSE  FALSE   FALSE   FALSE FALSE FALSE    FALSE
4   FALSE  FALSE   FALSE   FALSE FALSE FALSE    FALSE
5   FALSE   TRUE    TRUE   FALSE FALSE FALSE    FALSE
6   FALSE  FALSE   FALSE    TRUE FALSE FALSE    FALSE
```

From the above R code, we can see that the variables in the `movies` dataframe are:

- **Title** of the movie
- The **Year** the movie was released
- The movie's average **Rating** (out of 10), based on the number of **Votes** submitted
- The **Length** of the movie (in minutes)
- The movie **Budget** (this is in USD, as shown by **BudgetCurrency**, unless otherwise stated)
- The movie's rating as suggested by the Motion Picture Association of America (**mpaa**)
- Various logical variables indicating the movie's genre (e.g. **Animation**, **Horror** etc.)

For example, the movie "10 Cloverfield Lane" was released in 2016; it has an average rating of 7.2/10 from 226, 726 votes submitted to the Internet Movie Database (IMDB); the movie length is one hour 44 minutes; the budget was around US$ 15M; it was rated PG-13 by the Motion Picture Association of America; and it has been classified as a drama/horror/mystery movie.

Examples of querying the movies dataframe

1. Note that there are some missing values – for example, the budget for the original version of 101 Dalmatians, released in 1961, is unknown. What if we wanted to know how many movies had known budgets?

2. What if we wanted to create a new dataset with only those movies made since the year 1998?

3. Or (harder) movies that cost more than US$ 100,000 but are not R rated?

For the first query:

```
x1 = is.na(movies$Budget)          # Will return TRUE if the budget is NA
head(x1)                           # Compare to dataset above – looks good
 [1] FALSE  TRUE FALSE FALSE FALSE FALSE

x2 = !is.na(movies$Budget)         # Flip things round: NA = FALSE
head(x2)
 [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE

budget_known = movies$Budget[x2]   # Apply logical vector to the movie budgets
head(budget_known)
 [1] 1.05e+08 7.50e+07 8.50e+07 1.50e+07 3.00e+07 6.00e+06

length(budget_known)               # This will tell us how many movies have known budgets
 [1] 3990

length(movies$Budget[!is.na(movies$Budget)])      # A bit more concise!
 [1] 3990
```

Second query:

```
movies2 = subset(movies, Year>=1998)

dim(movies2)
 [1] 3478   23
```

And the last query:

```
movies3 = movies[movies$Budget>100000 & movies$mpaa != "R", ]
dim(movies3)
 [1] 3174   23
```

Example 2.2: Follow the steps at the start of Section 2.4.2 to load the `ncldata` package and to read in the `movies` dataframe. Then answer the following questions.

(a) How many people submitted votes for the films "*10 Things I Hate About You*", "*Midnight in the Garden of Good and Evil*" and "*Zoolander*"? **Answers:**_____

(b) How many films are classified as "Action"? How many are classified as "Romance"? How many "Short" films are there in the database? **Answers:**_____

(c) How many films were made between the Years 1970 and 2000 (inclusive)? **Answer:**_____

(d) How many movies are longer than two hours in duration? **Answer:**_____

(e) How many movies made since the year 2005 (not including 2005) are longer than three hours in duration? **Answer:**_____

Example 2.3: Load and explore the `music` dataset from the `ncldata` package by typing the following lines of code (Note: This will only work if you have correctly installed the `ncldata` package and read the package into your working RStudio space using the `library` command; see example 2.2 and Section 2.4.2):

```
data(music)
# What's this dataset all about? Let's find out
?music
```

(a) How many tracks do we have data for, and on how many variables? **Answers:**_____

(b) How many tracks do we have from *The Beatles* in this dataframe? **Answer:**_____ What about *Lady GaGa*? **Answer:**_____

(c) How many tracks have a tempo of at least 130 beats per minute? **Answer:**_____

(d) Of the tracks with a tempo of at least 130 beats per minute, how many have been released since the year 2000 (including the year 2000)? **Answer:**_____

(e) What is the most popular key for tracks in this database? **Answer:**_____

2.5 Graphical and numerical summaries

R has very powerful built-in functions for summarising datasets both graphically and numerically. Most of these are fairly intuitive, so in this section we provide only a brief review of material from stage 1.

For example, we might want to explore the distribution of budgets in the `movies` dataframe. As we know, there are some missing values in the dataset, but most of the built-in functions for data analysis in R have an argument we can invoke to ignore these:

```
# To find the mean of the movie budgets but ignoring the missing values
mean(movies$Budget, na.rm = TRUE)
 [1] 41245023

# And the standard deviation
sd(movies$Budget, na.rm = TRUE)
 [1] 68612118

# How about a range of numerical summaries with a single function?
summary(movies$Budget, na.rm = TRUE)
      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
0.000e+00 9.000e+06 2.500e+07 4.125e+07 5.000e+07 1.900e+09       998
```

From the listing above we can make a few observations:

- The mean budget is larger than the median: ∼US$ 41M, compared to US$ 25M. This indicates a positively skewed dataset, as the mean is being 'pulled up' by some very large budgets (of around USD$1.9Bn?! Which movie is this?! Must be amazing...)
- The standard deviation seems extremely large; again, this could be due to a small number of extremely large budgets...
- ... or perhaps some unrealistically *small* budgets (of US$ 0?!)

```
# Let's find out the position of the most expensive movie
which(movies$Budget == max(movies$Budget, na.rm=TRUE))
 [1] 1211

# What's the name of this movie?
movies$Title[1211]
 [1] "Enthiran"
# Never heard of it!

# Which movies were made for free?!
movies$Title[which(movies$Budget==0)]
 [1] "Homefront"      "Just Friends"   "Knock Knock"    "Now You See Me"
 [5] "Safe House"     "Snitch"
```

What about some graphical summaries? By far the most commonly-used graphics are histograms and boxplots. Of course, there are so many more graphical options available in R; you should consult your MAS1608 notes for a more comprehensive coverage.

```
# Partitions the plotting space: 1 row, 3 columns
par(mfrow = c(1,3))

# Now we'll try three different methods for producing histograms, based on class width/number of bars
# 1. Sturges' rule; the default, tries to find an optimal number of bars
hist(movies$Budget, col="wheat",breaks="sturges", xlab="Budget (USD)", main="Sturges' rule")
# 2. Freedman-Diaconis's rule; tries to optimise the class width
hist(movies$Budget, col="gold",breaks="fd", xlab="Budget (USD)", main="FD rule")
# 3. Scott's rule; also tries to optimise the class width
hist(movies$Budget ,breaks="scott", xlab="Budget (USD)", main="Scott's rule")
```

```
par(mfrow=c(1,2))
# Let's try a boxplot now
boxplot(movies$Budget, col="gold", main="Boxplot of movie budgets", horizontal=TRUE, xlab="Budget (USD)")

# Outliers are spoiling things a bit. Let's remove them
boxplot(movies$Budget, col="gold", main="Boxplot of movie budgets", horizontal=TRUE, xlab="Budget (USD)", outline=FALSE)
```

The histograms shown in Figure 1 reveal just how positively skewed the movie budget data are. They also show how care should be taken with the method chosen to produce the histogram. Of course, this is a subjective choice, and some experimentation might be necessary – *especially* with such a heavily skewed dataset. The boxplots also indicate just how skewed our dataset is; to get a handle on the distribution of most of of our data, it might be preferable to produce the boxplot with the outliers removed.

Table 3 gives a summary of some R commands for commonly-used tables and graphs.

Command	Comment	Example
`table`	Contingency table	`table(x)`
`barplot`	Generate a bar chart	`barplot(table(x))`
`hist`	Histogram	`hist(x)`
`plot`	Scatterplot	`plot(x, y)`
`points`	Add points to a plot	`points(x, y)`
`lines`	Add lines to a plot	`lines(x, y)`
`boxplot`	Box and whisker plot	`boxplot`

Table 3: Summary of R commands for commonly-used tables/graphs/graphical features

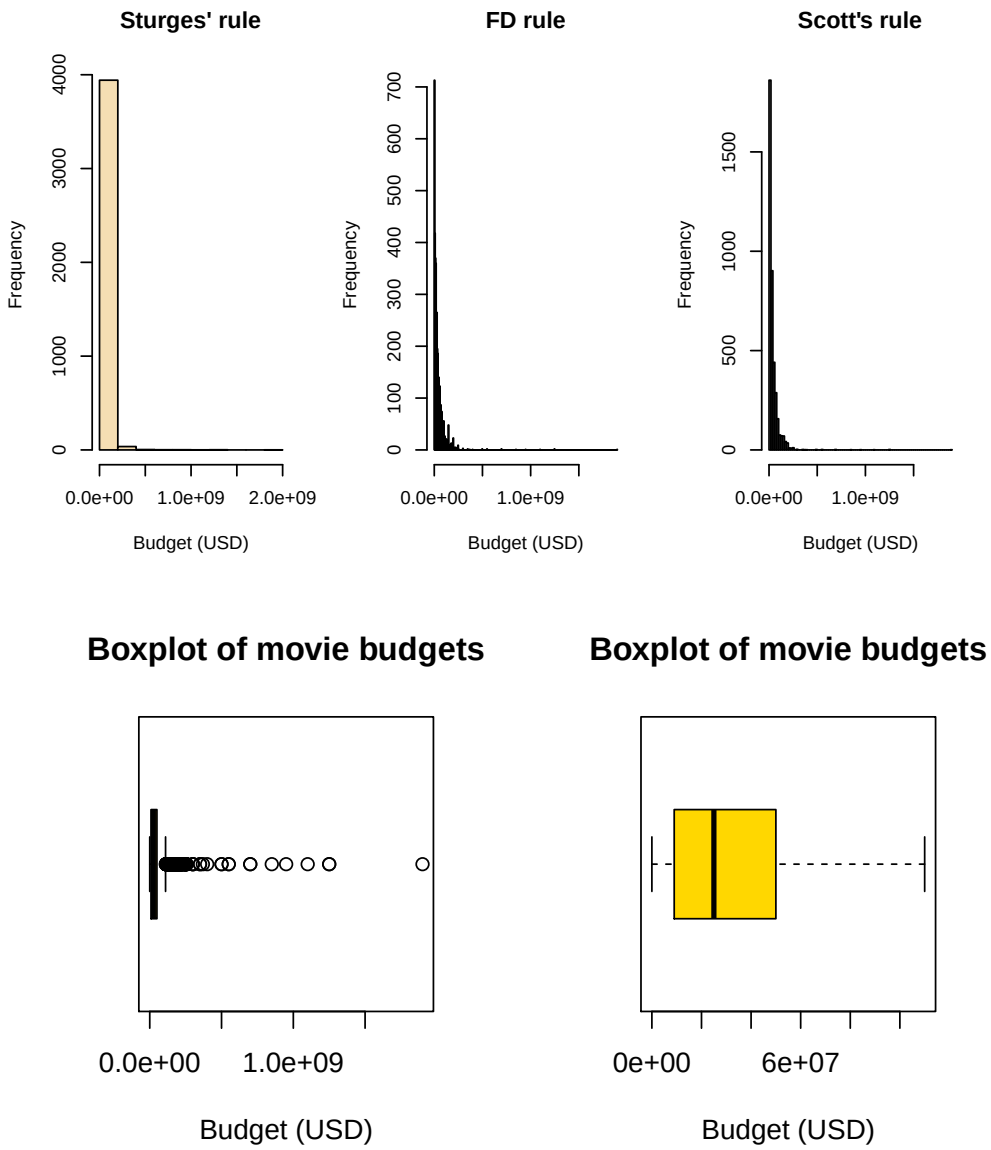


Figure 1: Histograms for the movie budgets, constructed using three different methods (top row); boxplots (bottom row), with outliers removed (right).

Example 2.4: Load the `music` dataset into RStudio, just as you did in Example 2.3.

(a) Find the mean and standard deviation for the duration of tracks in this dataset.

(b) Produce histograms for track duration and danceability; put these histograms on the same panel of plots, use different colours for the histograms (you can choose whichever colours you like!) and make sure you provide suitable titles and labels on the axes.

(c) You are asked to investigate whether or not the duration of tracks has changed over time.

(i) Use the `plot` command in R to produce a scatterplot of the year each track was released against track duration.

(ii) Try out the following code, and comment on what you see in the plots in part (c).

```
boxplot(music$duration~music$decade)
```

(d) On average, how do energy scores for "Hip Hop" tracks compare to "Electronic" tracks?

Example 2.5: The code in the following listing will download COVID-19 data from the website of the *European Centre for Disease Prevention and Control*. Note: this dataset contains information for 2019-2020 only.

```
covid = read.csv("https://opendata.ecdc.europa.eu/covid19/casedistribution/
csv", na.strings = "", fileEncoding = "UTF-8-BOM")
```

(a) Use the `head` command to take a look at the first 6 rows of the dataframe, to get a feel for the information contained in the dataset. Use the `dim` command to find out the size of the dataframe.

(b) We can use the `attach` command to directly access the columns in the dataframe by their headings. This might make it a bit easier to isolate data for certain countries. For example:

```
# Enable access to columns via their headings
attach(covid)

# Find the total number of cases, and deaths
sum(cases); sum(deaths)
 [1] 71503614
 [1] 1612833


# What about Great Britain?
sum(cases[countryterritoryCode=="GBR"], na.rm=TRUE)
 [1] 1849403
sum(deaths[countryterritoryCode=="GBR"], na.rm=TRUE)
 [1] 64170
```

```
1 # What about a plot of cases/deaths through time in Great Britain? See Figure 2
2 par(mfrow=c(1,2))
3 plot(rev(cases[countryterritoryCode=="GBR"]), type="l", xlab="Day", ylab=
    "Confirmed cases", main="GBR COVID-19 cases")
4 plot(rev(deaths[countryterritoryCode=="GBR"]), type="l", xlab="Day", ylab
    ="Confirmed deaths", main="GBR COVID-19 deaths")
```

(i) Find the total number of confirmed cases, and deaths, due to COVID-19, for three other countries of your choosing.

(ii) Why is it necessary to use the `rev` command in lines 3 and 4 in the above listing?

(iii) For the three other countries you chose in part (i), produce plots showing confirmed cases and deaths through time.

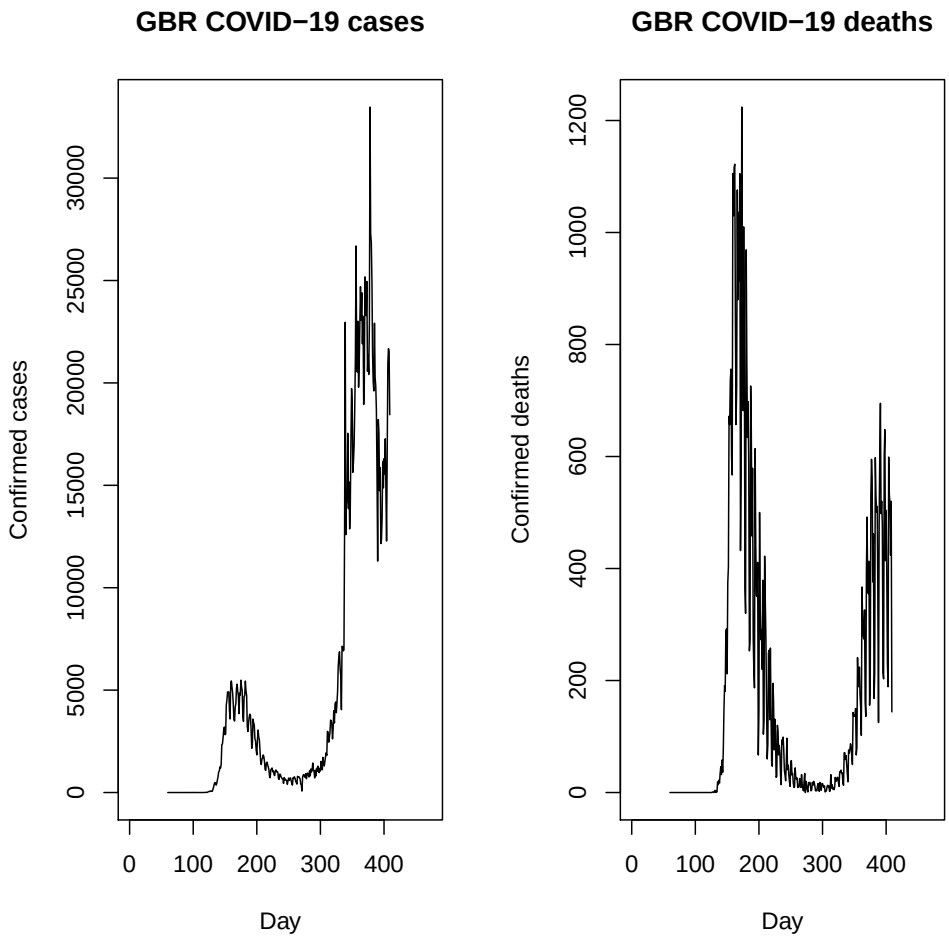


Figure 2: Daily cases and deaths due to COVID-19 in Great Britain.

2.6 Functions, for loops and if statements

The final part of this R Refresher will focus on what was probably the biggest part of the stage 1 R course: functions, for loops and if statements. In this section of the notes, we will focus on a single example that will pull all of these aspects of R programming together: How to fill a sticker album!

The 2018 *Panini* sticker album for the Fifa world cup football finals had a whopping 682 stickers to collect (including 40 awesome glitter stickers!). Each packet of stickers cost 80p and contained 5 stickers. The obvious question to ask is:

"How much does it cost to fill a Panini World Cup '18 sticker album?"

Now look at the code in the listing below.

```
# Create a vector for the sticker sequence - suppose for now that we only have 50 stickers
(stickers = seq(1, 50))     # Can actually be written more concisely as stickers = 1:50
  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 [25] 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 [49] 49 50

# Now let's buy a packet!
(packet = sample(1:50, 5, replace=TRUE)) # Random sample of size 5 from {1,2,...,50}
 [1] 10 23 20  48 11                                # We got stickers 10, 11, 20, 23 and 48!

# As a way of keeping track of the stickers we have collected, we could set the corresponding sticker
# sequence entry to zero
stickers[packet[1]] = 0      # Will set stickers[10] to 0
stickers[packet[2]] = 0      # Will set stickers[23] to 0
stickers[packet[3]] = 0      # Will set stickers[20] to 0
stickers[packet[4]] = 0      # Will set stickers[48] to 0
stickers[packet[5]] = 0      # Will set stickers[11] to 0

# That's a bit repetitive! We could put this inside a loop
for(j in 1:5){
  stickers[packet[j]] = 0}

stickers
  [1]  1  2  3  4  5  6  7  8  9  0  0 12 13 14 15 16 17 18 19  0 21 22  0 24
 [25] 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47  0
 [49] 49 50
```

After running the code in the above listing a few times, you should start to get a feel for what's going on. The aim is to keep buying packets of stickers and setting the corresponding entry in the `sticker` sequence to zero as a way of keeping track of the stickers we have collected. The code above adjusts this sequence after buying just a single packet of stickers. We need to do this repeatedly, an unknown (and probably very large!) number of times. So, another `for` loop is required.

```
# Re-set the sticker sequence
(stickers = seq(1, 50))
  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 [25] 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 [49] 49 50

# Now let's start a big for loop, in which we repeatedly buy packets of stickers
for(i in 1:10000){          # 10,000 is a big number - surely this will be enough packets?
  packet = sample(1:50, 5, replace=TRUE)

  for(j in 1:5){
    stickers[packet[j]] = 0}# Close the j loop
    }# Close the i loop
stickers
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [37] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Looking at the listing above, it looks like we've filled the sticker album after buying 10,000 packets of stickers – notice that all entries in the sticker sequence vector are zero. But, we don't know exactly when this happened. Did it happen after we bought the 10,000th packet? Or after we bough the 200th packet? Or the 1524th packet?! We need to 'break' the code as soon as all entries in the sticker sequence are zero.

```
(stickers = seq(1, 50))
  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 [25] 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 [49] 49 50

for(i in 1:10000){
  packet = sample(1:50, 5, replace=TRUE)

  for(j in 1:5){
    stickers[packet[j]] = 0}

# Now inside the for loop use an if statement to break the code when all entries in the sticker sequence are zero
  if(sum(stickers)==0) {
    break}
  }

# When did the code break?
i
 [1] 37
```

So this time it only took 37 packets of stickers to completely fill the album! Of course, in this toy example we only have 50 stickers to collect. For the *Panini* football sticker album, there are 682!

Really, what we'd like to do is run the code in the above listing repeatedly, to see what happens for different 'people' trying to fill their sticker albums. It might be a good idea to put all of this into a `function` now.

```
1  FillAlbum = function(N, k){          # N = something large, k = number of stickers to collect
2    stickers = seq(1, k)
3
4    for(i in 1:N){
5    packet = sample(1:k, 5, replace=TRUE)
6
7    for(j in 1:5){
8      stickers[packet[j]] = 0}
9
10   if(sum(stickers)==0) {
11     break}
12   }
13   return(i)}
```

Notice, from the code listing above, the general structure of a function in R:

- The word `function` (line 1), followed by round brackets and some "arguments" to be passed through the function (arguments that we must specify when we execute the function)
- An appropriate name for the function (`FillAlbum` here, see line 1)
- The use of braces surrounding the operations within the function (lines 1 and 13; all other braces relate to the for loops/if statement)
- A return statement (line 13), declaring what we would like the function to spit out

We can now execute our `FillAlbum` function to see how many packets of stickers different "people" might need to buy in order to fill their albums.

```
# Let's try for an album containing just 50 stickers again
FillAlbum(10000, 50)
 [1] 52
FillAlbum(10000, 50)
 [1] 32
FillAlbum(10000, 50)
 [1] 40

# Now let's try for the full album of 682 stickers!
FillAlbum(10000, 682)
 [1] 1290
FillAlbum(10000, 682)
 [1] 1611
FillAlbum(10000, 682)
 [1] 1252
```

Referring to the code in the listing above, we have simulated three people trying to fill the full *Panini* sticker album. Respectively, they needed to buy 1,290, 1,611 and 1,252 packets of stickers to do this. We could now execute this function many times, within another `for` loop.

```r
# Let's simulate 1000 people trying to fill their albums
packets_needed = vector("numeric", 1000)

for(k in 1:1000){
  packets_needed[k] = FillAlbum(10000, 682)}

# Each packet costs 80p
cost = packets_needed*0.8

mean(packets_needed); mean(cost)
 [1] 974.609
 [1] 779.6872

sd(packets_needed); sd(cost)
 [1] 181.351
 [1] 145.0808

par(mfrow=c(1,2))
hist(packets_needed, col="misty rose", xlab="Packets", main="Packets needed")
hist(cost, col="chocolate", xlab="Pounds",main="Cost")
```

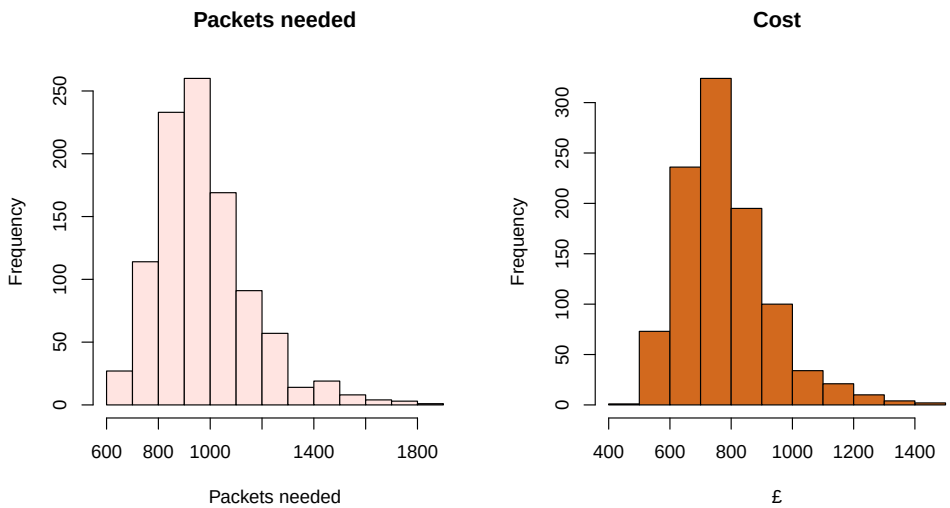


Figure 3: Histograms showing the number of packets of stickers needed to fill the *Panini* football sticker album (left), and the associated cost (right).

This analysis is really insightful! We can see that, on average, it takes around 975 packets of stickers to fill the album (standard deviation 181), with an associated cost of around £780 (standard deviation £145). The histograms in Figure 3 reveal the full distribution for the number of packets needed and the associated cost. The larger the number of people simulated (we used 1,000 here), the more accurate this distribution will be!

3 Simulating Random Variables

3.1 Discrete distributions: Binomial, geometric and Poisson

The binomial distribution: If $X \sim Bin(n, p)$ then X has PMF (probability mass function) given by:

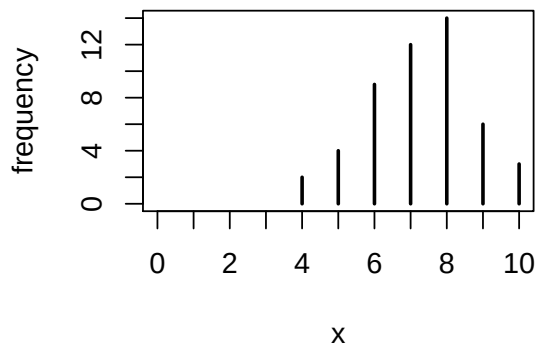
There is no closed formula for the CDF (cumulative distribution function; i.e. $\Pr(X \leq x)$).

The R commands `dbinom`, `pbinom`, and `rbinom` are used respectively to compute the PMF, compute the CDF, and simulate from a binomial distribution. Here are examples:

```
dbinom(5, 10, 0.7)        # Pr(X = 5), X ~ Bin(10, 0.7)
pbinom(4, 10, 0.7)        # Pr(X ≤ 4), X ~ Bin(10, 0.7)
rbinom(50, 10, 0.7)       # Sample a Bin(10, 0.7) distribution 50 times
```

Here's an example of how to create a bar plot of a sample from a binomial distribution:

```
x = rbinom(50, 10, 0.7)
plot(table(x), xlim = c(0,10), xaxt = 'n', xlab = 'x', ylab = 'frequency')
axis(1, at = seq(0, 10))
```



The geometric distribution: If $Y \sim Geom(p)$ then Y has PMF and CDF given by:

Note that Y takes values in $1, 2, 3, \ldots$. R uses a slightly different definition of the geometric distribution. We use the definition that Y is the number of Bernoulli trials with parameter p up to and including the first trial which is a success. The definition in R is the number of trials up to, but **not** including the first success, so in R geometric random variables take values $0, 1, 2, \ldots$. Like the binomial distribution, the R commands `dgeom`, `pgeom`, and `rgeom` are used respectively to compute the PDF, compute the CDF, and simulate from the geometric distribution. However, they need to be adjusted to match our definition:

```
dgeom(4, 0.2)          # Pr(Y = 5), Y ~ Geom(0.2)
pgeom(2, 0.2)          # Pr(Y ≤ 3), Y ~ Geom(0.2)
1 + rgeom(100, 0.2)    # Sample a Geom(0.2) distribution 100 times
```

Here's a function to replace `dgeom` with our definition of the geometric distribution:

```
mydgeom = function(x, p) {
    dgeom(x-1, p)}
```

The Poisson distribution: If $Z \sim Po(\lambda)$ then Z has PMF given by:

There is no closed formula for the CDF.

Examples of R commands:

```
dpois(5, 3.5)      # Pr(Z = 5), Z ~ Po(3.5)
ppois(2, 3.5)      # Pr(Z ≤ 2), Z ~ Po(3.5)
rpois(100, 3.5)    # Sample a Po(3.5) distribution 100 times
```

Summary:

Distribution	Binomial	Poisson	Geometric ☠
PMF	`dbinom(...)`	`dpois(...)`	`dgeom(...)`
CDF	`pbinom(...)`	`ppois(...)`	`pgeom(...)`
sample	`rbinom(...)`	`rpois(...)`	`rgeom(...)`

3.2 Continuous random variables

R has functions for the uniform, exponential and normal distributions, where we now refer to the probability *density* function, rather than probability *mass* function:

Distribution	Uniform	Exponential	Normal
PDF	`dunif(...)`	`dexp(...)`	`dnorm(...)`
CDF	`punif(...)`	`pexp(...)`	`pnorm(...)`
Quantile	`qunif(...)`	`qexp(...)`	`qnorm(...)`
sample	`runif(...)`	`rexp(...)`	`rnorm(...)`

Examples of R commands:

```
dunif(3, 2, 5)      # f_X(3), X ~ U(2,5)
pexp(1.5, 5)        # F_Y(1.5), Y ~ Exp(5)
rnorm(10, 0, 2)     # Sample a N(0,2^2) distribution 10 times
```

For the standard uniform ($U(0,1)$) and standard normal ($N(0,1)$) distributions you don't need to provide the parameters $a = 0$, $b = 1$ and $\mu = 0$, $\sigma = 1$ respectively. For example:

```
runif(20)        # Samples a U(0,1) distribution 20 times
pnorm(1.96)      # Pr(Z < 1.96), Z ~ N(0,1)
[1] 0.9750021
```

Quantile functions such as `qexp` and `qnorm` solve equations of the form $F_X(\alpha) = p$, for α, given a probability p. For example:

```
qnorm(0.9750021)
[1] 1.96
```

Example 3.1: Suppose annual maximum wave heights observed off the coast at a flood-prone town are assumed Normally distributed, with mean 2 metres and standard deviation 0.5 metres.

(a) Write down the R command to find the probability that the largest wave height next year will exceed 3.25 metres. **Answer:** _____

(b) Write down the R command to estimate the height of a new sea wall such that we might expect the town to be flooded, on average, once per century. Why might our modelling assumption be invalid? **Answer:** _____

Example 3.2: Use R to generate a sample of size 10,000 from an exponential distribution with rate 2.5, and store this sample in the vector x.

(a) Use your vector x to estimate $\Pr(X \leq 0.5)$, where $X \sim Exp(2.5)$.
Answer: _____

(b) How does your estimate in part (a) compare to the true probability? Use R here – do not perform any calculations by hand. **Answer:** _____

Example 3.3: Write down the R code, and corresponding result, to answer each of the questions below. The first one has been done for you.

Model	Question	R code	Answer
$X \sim Po(3)$	Find $\Pr(X = 6)$.	`dpois(6,3)`	0.0504
	Find $\Pr(X < 5)$.		
	Generate 3 observations.		
$Y \sim Bin(10, 0.2)$	Find $\Pr(Y \leq 4)$.		
	Find $\Pr(Y \geq 4)$.		
	Find y such that $\Pr(Y \leq y) = 0.677$.		
$Z \sim N(0, 1)$	Find $\Pr(Z \leq 0.2)$.		
	Find $\Pr(Z > -2.5)$.		
	Generate 3 observations.		

3.3 Some more advanced probability models

The main reason to use computers for Mathematics and Statistics is to solve problems which cannot be done by hand or which require very lengthy calculations. This section contains a simple motivating example to illustrate this.

The number of arrivals per day at a university IT help-desk can be modelled using a Poisson distribution. However, it is thought that the mean of the Poisson distribution might vary from day-to-day. For example, the desk is often busier nearer the end of term due to more students having deadlines. The following model is proposed. Suppose the number of arrivals $X|\Lambda \sim Po(\Lambda)$ where the rate Λ is itself a random variable, with $\Lambda \sim Exp(c)$ for a constant $c = 0.05$. What can we say about the distribution of X?

The random variable X does not have a distribution which we can immediately recognize. In fact, much can be done analytically, but we will use simulation to answer the following questions:

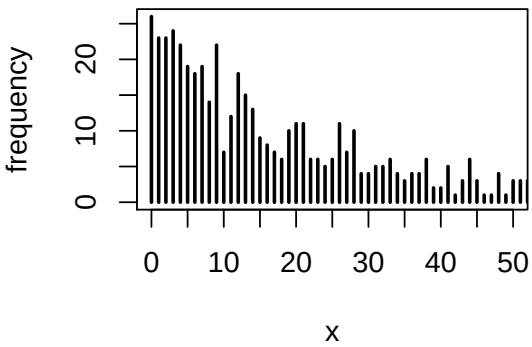
1. What are the expectation and variance of X?
2. What is $\Pr(X > 30)$?

Here is code to simulate a number n of samples from the **distribution** of X:

```
arrivals = function(n, c = 0.05) {
    x = vector(mode = 'numeric',length = n)
    for (i in 1:n) {
        lambda = rexp(1, rate = c)
        x[i] = rpois(1, lambda)
    }
    return(x)
}
```

Here's how to create a bar plot of the sample and compute the **sample** mean and variance:

```
x = arrivals(500)
plot(table(x), xlim = c(0, 50), xaxt = 'n', xlab = 'x', ylab = 'frequency')
axis(1, at = seq(0, 50, 5))
```




```
mean(x); var(x)
[1] 19.618
[1] 382.1764
```

The mean and variance of the distribution of X will be close to the sample mean and variance that we see in the output above. To calculate $\Pr(X > 30)$ we calculate the proportion of times this occurs in the sample:

```
sum(x>30)/500
[1] 0.202
```

The bigger the value we use for n, the closer the sample distribution will be to the true distribution for X; and the better our estimates of the mean, variance and probabilities will be.

3.4 Mixtures of normals

Suppose $\mu_1, \mu_2, \sigma_1 > 0, \sigma_2 > 0$ are fixed constants, and w_1, w_2 are positive constants with $w_1 + w_2 = 1$. Consider the following function:

$$f(x) = w_1 f_1(x) + w_2 f_2(x)$$

where f_1 and f_2 are the density functions for $Z_1 \sim N(\mu_1, \sigma_1^2)$ and $Z_2 \sim N(\mu_2, \sigma_2^2)$ respectively.

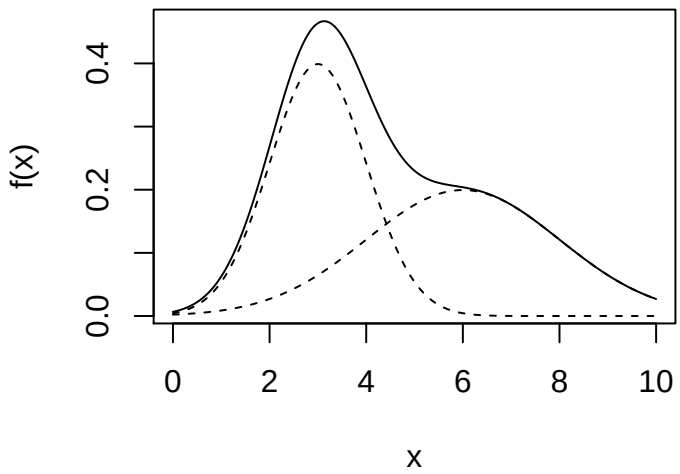
Check that $f(x)$ represents a valid probability density function:

If a random variable X has PDF corresponding to $f(x)$ we say it is a **mixture** of normal distributions $N(\mu_1, \sigma_1^2)$ and $N(\mu_2, \sigma_2^2)$ with weights w_1, w_2. Note that X is **not** the sum of two normal random variables, i.e.

$$X \neq w_1 Z_1 + w_2 Z_2 \quad \text{where } Z_i \sim N(\mu_i, \sigma_i^2),\ i = 1, 2.$$

The right hand side is actually another normal distribution (as seen in MAS1608), but X **does not have** the PDF of a normal random variable.

For example: suppose $\mu_1 = 3, \sigma_1 = 1$ and $\mu_2 = 6, \sigma_2 = 2$ with $w_1 = w_2 = 1/2$. Here is a plot of $f(x)$:



Consider how to sample X from this distribution:

1. Sample a random variable $J \in \{1, 2\}$ such that $\Pr(J = 1) = w_1$ and $\Pr(J = 2) = w_2$. The random variable J tells you which **component** of the mixture to sample X from.

2. If $J = 1$ then sample X from $N(\mu_1, \sigma_1^2)$, but if $J = 2$ then sample X from $N(\mu_2, \sigma_2^2)$.

Here is an R function to perform this sampling n times:

```
normal.mixture = function(n, mu1, sig1, w1, mu2, sig2, w2) {
    p = c(w1, w2)
    x = vector(mode = 'numeric', length = n)
    for (i in 1:n) {
        j = sample(c(1, 2), 1, prob = p)
        if (j==1) {
            x[i] = rnorm(1, mu1, sig1)
        }
        else {
            x[i] = rnorm(1, mu2, sig2)
        }
    }
    return(x)
}
```

Write down any notes about this function.

Normal mixture distributions can contain more than two components – the generalization is very straightforward. They offer a very flexible way to model data with different shaped distributions.

Example 3.4: Investigations into infants suffering from *severe idiopathic respiratory distress syndrome* have shown that whether the infant survives may be related to their weight at birth. Suppose that you are interested in developing a distribution for the birth weight X of such infants. You have a normal $N(2.3, 0.52^2)$ distribution for the birth weight (in kg) of infants who survive and a normal $N(1.7, 0.66^2)$ distribution for infants who die.

(a) If you believe that the proportion of infants that survive is 0.6, write down a suitable Normal mixture distribution for X.

(b) Produce some R code to sample from the normal mixture distribution you propose in part (a).

(c) Use your code in part (b) to simulate a sample of size 1000 from your normal mixture distribution, and produce a histogram of your sample.

4 The Inverse CDF Method

Let $U \sim U(0,1)$. Suppose $F(x)$ is a well-defined CDF which is invertible. Then the random variable $X = F^{-1}(U)$ has CDF $F(x)$.

Proof:

Example 4.1: Suppose we obtain two observations from a $U(0,1)$ distribution: 0.1 and 0.85. Use these values to obtain two observations from $X \sim Exp(2)$.

Example 4.2: Write some R code to sample n observations from an $Exp(2)$ distribution. Do not use the built-in `rexp` function. Check your code by comparing a histogram of some simulated realisations against the PDF for an $Exp(2)$ distribution. Also check your code against results from the built-in `rexp` function.

Example 4.3: The *Pareto distribution*, named after the Italian civil engineer, economist, and sociologist Vilfredo Pareto, is a power-law probability distribution that is used in the description of social, scientific, geophysical, actuarial, and many other types of observable phenomena. It was originally used to describe the distribution of wealth in a society, fitting the trend that a large portion of wealth is held by a small fraction of the population.

If $X \sim Pareto(\sigma)$,then it has cumulative distribution function

$$F_X(x;\sigma) = 1 - (\sigma/x)^{\log_4 5}, \quad x \geq \sigma, \sigma > 0.$$

Write a function in R to produce realisations from this distribution, and test your function out for $\sigma = \{1, 2, 5\}$.

Example 4.4: The *Weibull distribution* is often used in survival data analysis to model lifetimes (e.g. the lifetimes of patients receiving a particular drug). If $X \sim Weibull(\lambda, \kappa)$, then it has cumulative distribution function

$$F_X(x;\lambda,\kappa) = 1 - e^{-(x/\lambda)^\kappa}, \quad x \geq 0, \lambda, \kappa > 0.$$

Write a function in R to produce realisations from this distribution, and test your function out (a recent study suggested that, for a particular cancer drug, $\lambda = 2$ and $\kappa = 1.5$ for survival times in months after starting this drug).

Example 4.5: Patients arriving at a doctor's surgery have appointment times 15 minutes apart. The random variable L measures how late a patient is for his/her appointment, with negative values denoting early arrivals. The PDF of L is

$$f_L(\ell) = \begin{cases} 0.0025(\ell + 20), & -20 \leq \ell \leq 0, \\ 0.05e^{-\ell/10}, & \ell > 0. \end{cases}$$

(a) Sketch $f_L(\ell)$.

(b) Show that the CDF of L is

$$F_L(\ell) = \begin{cases} 0, & \ell < -20, \\ 0.00125(\ell + 20)^2, & -20 \leq \ell \leq 0, \\ 1 - \frac{e^{-\ell/10}}{2}, & \ell > 0. \end{cases}$$

(c) Suppose that 0.205 and 0.713 are two realized values of $U \sim U(0, 1)$. Show that the corresponding realized values of L are -7.19 and 5.55 under the inverse CDF simulation method.

5 Transformation of Random Variables

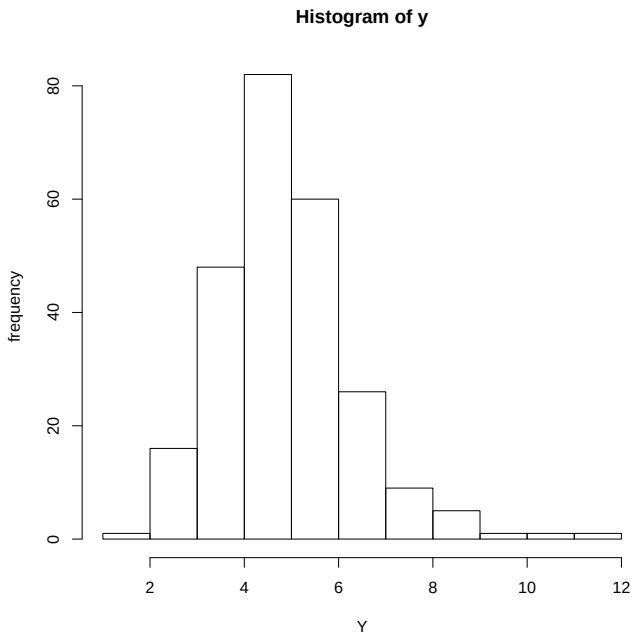
5.1 Transforming a normal random variable

The normal distribution is very widely used to model data. However, normal random variables take values on the entirety of $\mathbb{R}$ and they are symmetric around the mean. This is unrealistic in many situations – for example, if we want to model people's heights or weights. One way to obtain a positive random variable is to define

$$Y = e^X, \quad \text{where } X \sim N(\mu, \sigma^2).$$

Then Y is a random variable which is a **transformation** of X. What does the distribution of Y look like? We can use sampling to find out, at least approximately:

```
mu = 1.5
sigma = 0.3
n = 250
y = exp(rnorm(n, mean = mu, sd = sigma))
hist(y, xlab = 'y', ylab = 'frequency', main = 'Histogram of Y')
```



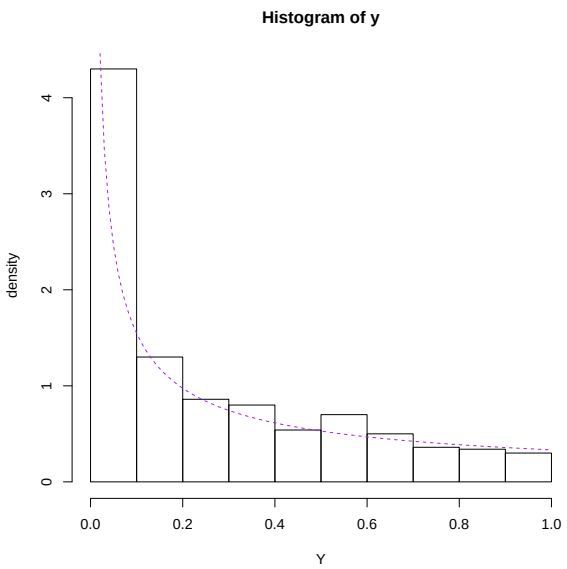
The random variable Y can only adopt positive values, and the distribution is positively skewed (long tail on the right). This kind of distribution is useful to model quantities like height – but what is the PDF of Y? A first guess is $f_Y(y) = f_X(\log(y))$ since log maps Y back to X, but this is **not** the correct answer.

5.2 Computing PDFs for transformed random variables

Example 5.1: If $X \sim U(0,1)$, what is the PDF of $Y = X^3$?

To confirm this result, we can simulate Y. Does the histogram look right?

```
y = (runif(500))^3
hist(y, xlab = 'Y', ylab = 'density', main="", prob=TRUE)
curve((1/3)*(x^(-2/3)), from=0, to=1, col="purple", lty=2, add=TRUE)
```



Notes:

Example 5.2: If $X \sim N(0, 1)$, what is the PDF of $Y = X^2$?

Example 5.3: If $X \sim N(\mu, \sigma^2)$, what is the PDF of $Y = e^X$? This is the example used at the start of this section. We say Y has the **log-normal** distribution and write $Y \sim LN(\mu, \sigma^2)$.

5.3 The transformation method

These examples are all special cases of the following result.

If X has PDF $f_X(x)$ and $h(x)$ is a monotonically increasing or monotonically decreasing function then h is invertible and the PDF of $Y = h(X)$ is

$$f_Y(y) = f_X(h^{-1}(y)) \times \left| \frac{d\left(h^{-1}(y)\right)}{dy} \right|$$

when y is in the range of $h(x)$ and zero otherwise.

This result allows direct calculation of $f_Y(y)$ and is called the **univariate transformation method**. We will see this result again in MAS2903.

Proof:

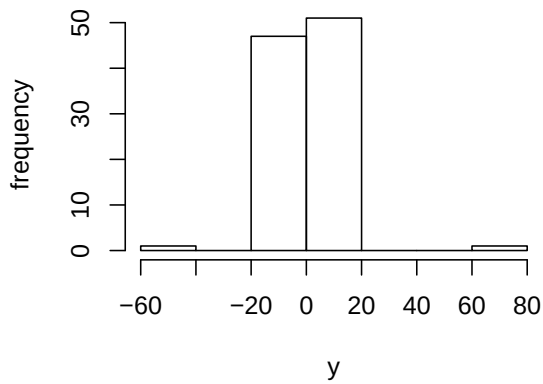
Example 5.4: When $X \sim U(0, 1)$, use the transformation method to find the PDF of $Y = X^3$.

Example 5.5: When $X \sim U(-1, 1)$ and $Y = e^X$, use the transformation method to find the PDF of Y.

Example 5.6: Use the transformation method to find the PDF of the log-normal distribution.

Example 5.7: Suppose $X \sim U(-\frac{\pi}{2}, \frac{\pi}{2})$ and $Y = \tan(X)$. Use simulation to study the distribution of Y and explicitly compute the PDF.

```
y = tan(runif(100, -pi/2, pi/2))
hist(y, xlab = 'y', ylab = 'frequency', main = '')
```



This distribution looks odd! Calculate the PDF:-

6 Bivariate distributions

In this section we consider general simulation from a bivariate distribution, as well as simulation from the bivariate Normal distribution.

6.1 Sampling from bivariate distributions

Suppose we have a pair of continuous random variables (X, Y) with joint PDF $f_{X,Y}(x, y)$. Recall that

$$f_{X,Y}(x, y) = f_X(x) f_{Y|X}(y \mid X = x)$$

where $f_X(x)$ is the **marginal** PDF of X and $f_{Y|X}(y \mid X = x)$ is the **conditional** PDF of Y given $X = x$. One way to simulate realizations of the pair (X, Y) is:

1. Simulate a realization x of X from the marginal distribution of X.
2. Simulate a realization y of Y from the conditional distribution of Y given $X = x$.

In the **trivariate case** we can add a third step: simulate Z given $Y = y$ and $X = x$.

Example 6.1: Suppose $X \sim U(0.1, 0.5)$ and

$$f_{Y|X}(Y \mid X = x) = \begin{cases} xe^{-xy}, & \text{when } y \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Write an R function to generate a $2 \times n$ **matrix** of samples from this distribution, and produce a scatter plot.

Note that, given that $X = x$, Y has an exponential distribution with parameter $\lambda = X$:

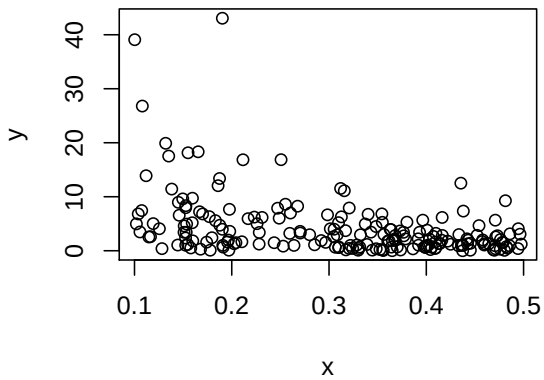
$$Y \mid X = x \sim Exp(x).$$

```
example6.1 = function(n) {
    output = matrix(0, 2, n)
    x = runif(n, 0.1, 0.5)
    y = rexp(n, rate = x)
    output[1, ] = x
    output[2, ] = y
    return(output)
}
```

Make some notes about the R function.

To obtain a scatterplot:

```
test = example6.1(200)
plot(test[1,], test[2,], xlab = 'x', ylab = 'y')
```



Example 6.2: (Trivariate case.) Suppose that $X \sim U(-1, 1)$ and $Y \sim Exp(\lambda)$ independently of X and that

$$f_{Z|X,Y}(Z \mid X = x, Y = y) = \frac{1}{\sqrt{2\pi}y} e^{-\frac{1}{2}\left(\frac{z-x}{y}\right)^2}.$$

Write an R function to sample realizations (X, Y, Z) in the case $\lambda = 5$ and produce a scatterplot of (X, Z). What would the plot look like if $\lambda = 1$?

Note that, given that $X = x$, $Y = y$ then Z has a normal distribution:

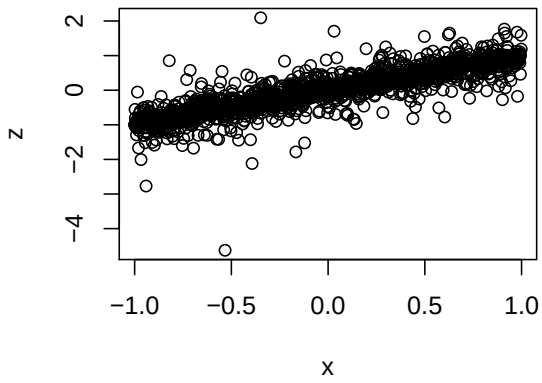
$$Z \mid X = x, Y = y \sim N(x, y^2).$$

```
example6.2 = function(n) {
   output = matrix(0, 3, n)
    x = runif(n, -1, 1)
    y = rexp(n, rate = 5)
    z = rnorm(n, mean = x, sd = y)
    output[1, ] = x
    output[2, ] = y
    output[3, ] = z
    return(output)
}
```

Make some notes about the R function.

To obtain a scatterplot:

```
output = example6.2(2000)
plot(output[1,], output[3,], xlab = 'x', ylab = 'z')
```



6.2 The multivariate normal distribution

We met the bivariate normal distribution in MAS1608. The bivariate normal distribution is a generalization of the normal distribution to pairs of random variables, or equivalently, to a distribution on vectors in $\mathbb{R}^2$. The multivariate normal distribution is the analogous distribution on vectors in $\mathbb{R}^n$.

6.2.1 The bivariate normal distribution

Suppose we have constants μ_x, μ_y, σ_x, σ_y and ρ such that $\sigma_x \geq 0$, $\sigma_y \geq 0$ and $-1 \leq \rho \leq 1$. Define the 2×2 matrix Σ by

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}.$$

Then define a joint probability density function by

$$f_{X,Y}(x,y) = \frac{1}{2\pi\sqrt{\det\Sigma}} \exp\left(-\frac{1}{2}Q(x,y)\right) \tag{1}$$

where

$$Q(x,y) = (\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu}) \tag{2}$$

and the column vectors $\underline{x}, \underline{\mu}$ are defined by $\underline{x} = (x,y)^T$, $\underline{\mu} = (\mu_x, \mu_y)^T$. If random variables (X,Y) have joint probability density given by $f_{X,Y}$ above, then we say that (X,Y) have a **bivariate normal distribution** and write

$$(X,Y)^T \sim N_2(\underline{\mu}, \Sigma).$$

Some of the terms in the definition can be expanded:

$$\sqrt{\det\Sigma} = \sigma_x\sigma_y\sqrt{1-\rho^2}$$

and

$$Q(x,y) = \frac{1}{\sigma_x^2\sigma_y^2(1-\rho^2)} \begin{pmatrix} x-\mu_x & y-\mu_y \end{pmatrix} \begin{pmatrix} \sigma_y^2 & -\rho\sigma_x\sigma_y \\ -\rho\sigma_x\sigma_y & \sigma_x^2 \end{pmatrix} \begin{pmatrix} x-\mu_x \\ y-\mu_y \end{pmatrix}$$
$$= \frac{1}{1-\rho^2}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 + \left(\frac{y-\mu_y}{\sigma_y}\right)^2 - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}\right].$$

However, equations (1) and (2) are **much** more important than the expansions just given.

It can be proved that the function $f_{X,Y}(x,y)$ in equation (1) integrates to 1 and therefore defines a valid joint PDF. The proof is omitted.

Remarks:

1. The vector $\underline{\mu} = (\mu_x, \mu_y)^T$ is called the **mean vector** and the matrix Σ is called the **covariance** matrix (or sometimes **variance-covariance** matrix).

2. Functions of the form $F(\underline{x}) = \underline{x}^T\Sigma^{-1}\underline{x}$ are called **quadratic forms**. Quadratic forms are functions $\mathbb{R}^n \to \mathbb{R}$ which satisfy certain properties. They crop up in several areas of mathematics and statistics.

3. The matrix Σ and its inverse Σ^{-1} are **positive definite**. A matrix A is positive definite if

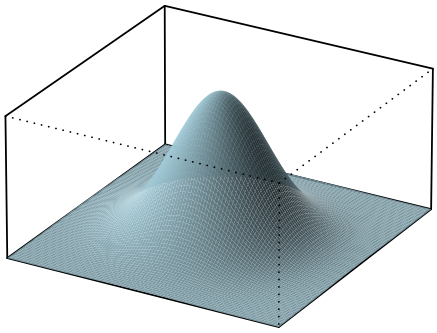
$$\underline{x}^T A\underline{x} \geq 0$$

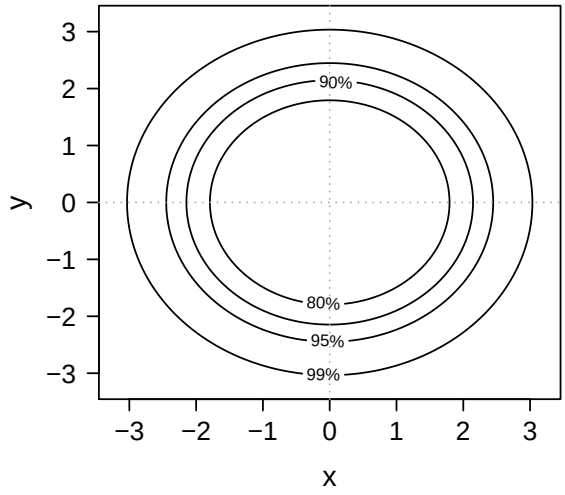
   for all non-zero vectors $\underline{x}$.

4. It follows that when $\mu_x = \mu_y = 0$, $Q(x,y)$ is of positive definite quadratic form.

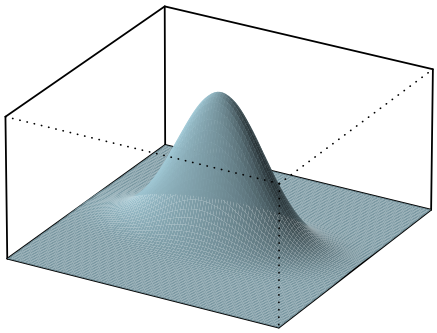
Over the next two pages we will visualize the bivariate Normal distribution using 3-D plots and contour plots.

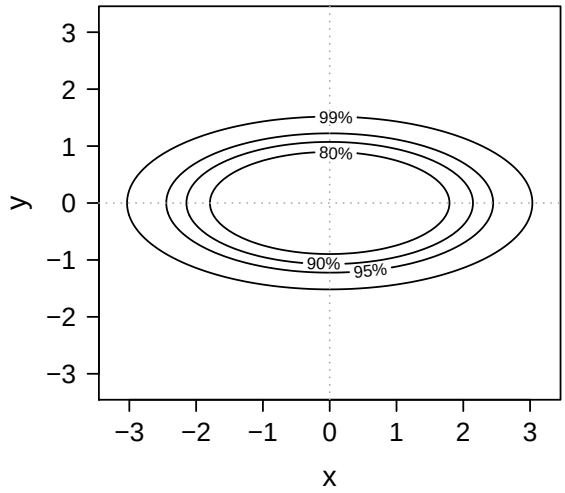
$\sigma_x = \sigma_y, \ \rho = 0$



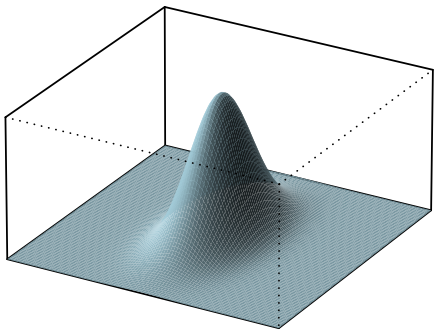


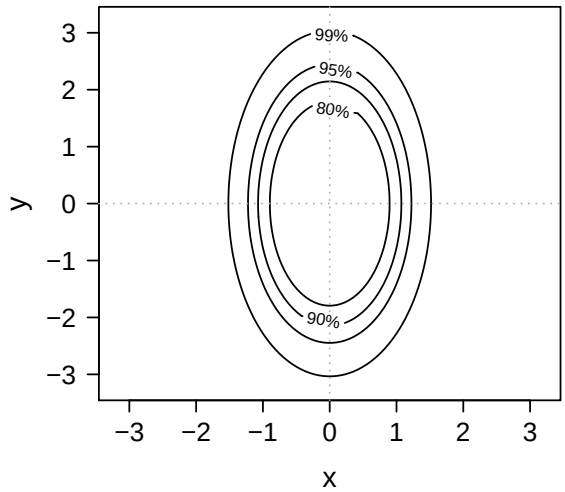
$\sigma_x = 2\sigma_y, \ \rho = 0$



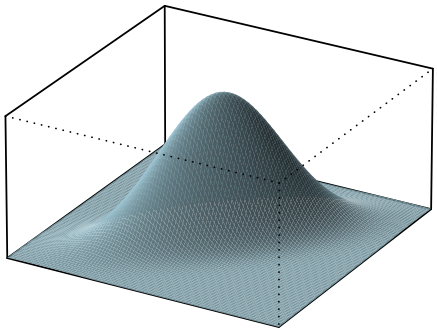


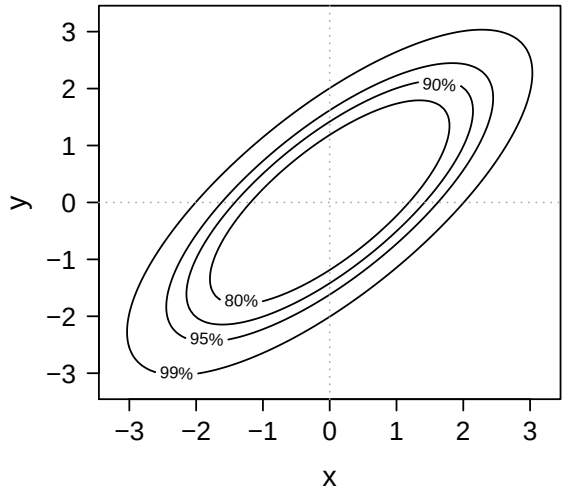
$2\sigma_x = \sigma_y, \ \rho = 0$



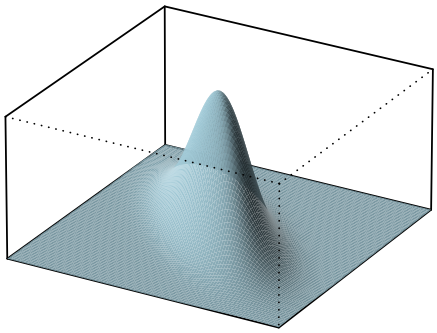


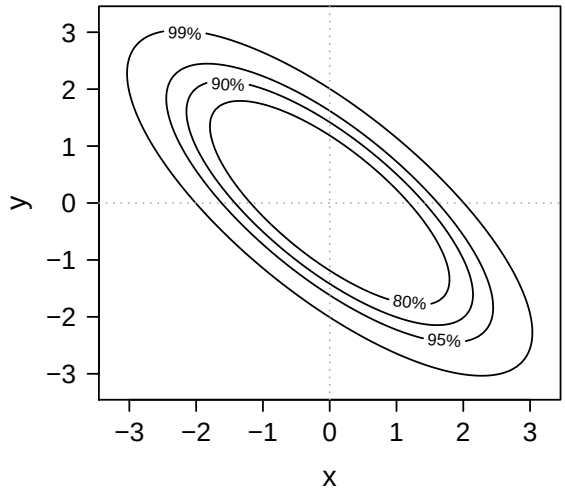
$\sigma_x = \sigma_y,\ \rho = 0.75$



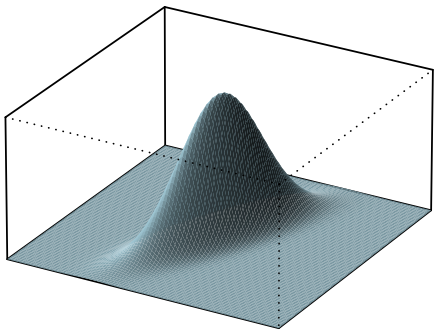


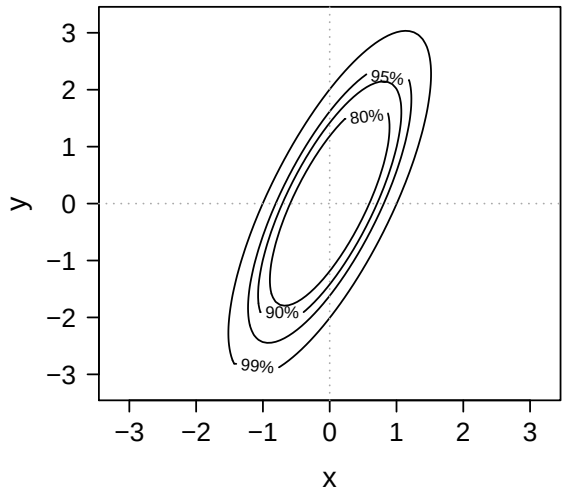
$\sigma_x = \sigma_y,\ \rho = -0.75$





$2\sigma_x = \sigma_y,\ \rho = 0.75$





The previous two pages show contour and surface plots for the density function on the xy-plane for different parameter values. Comments:

1. $Q(x, y) \geq 0$ with equality only when $\underline{x} = \underline{\mu}$. It follows that the density function has its mode at $\underline{x} = \underline{\mu}$. The above plots all have $\mu_x = \mu_y = 0$.
2. Changing the values of μ_x, μ_y does not change the shape of the plots, but corresponds to a translation of the xy-plane i.e. changing μ_x, μ_y just shifts the contours / surface to a new mode position.
3. The contours of equal density are **circular** when $\sigma_x = \sigma_y$ and $\rho = 0$ and **elliptical** when $\sigma_x \neq \sigma_y$ or $\rho \neq 0$.
4. σ_x and σ_y control the extent to which the distribution is **dispersed**.
5. The parameter ρ is the **correlation** of X, Y i.e. $\mathrm{Cor}\,(X, Y) = \rho$. (The proof is omitted.) Thus for non-zero ρ, the contours are at an angle to the axes.

When (X, Y) has a bivariate normal distribution then given a general set $A \subseteq \mathbb{R}^2$, calculation of the probability $\Pr((X, Y) \in A)$ requires a computer. However, the marginal and conditional distributions of the bivariate normal distribution are particularly simple – one of the reasons the distribution is so useful in statistics.

Suppose $(X, Y)^T \sim N_2(\underline{\mu}, \Sigma)$. The following results hold.

1. The marginal distributions are normal:
$$X \sim N(\mu_x, \sigma_x^2) \quad \text{and} \quad Y \sim N(\mu_y, \sigma_y^2).$$
2. The conditional distributions are normal:
$$\begin{aligned} X|Y = y &\sim N(\mu_x + \rho\frac{\sigma_x}{\sigma_y}(y - \mu_y), \sigma_x^2(1 - \rho^2)) \quad \text{and} \\ Y|X = x &\sim N(\mu_y + \rho\frac{\sigma_y}{\sigma_x}(x - \mu_x), \sigma_y^2(1 - \rho^2)). \end{aligned}$$
3. When $\rho = 0$, X and Y are independent.
4. Linear combinations of X and Y are also normally distributed:
$$aX + bY \sim N(a\mu_x + b\mu_y, a^2\sigma_x^2 + b^2\sigma_y^2 + 2ab\rho\sigma_x\sigma_y)$$
where a, b are constants. Thus a **linear combination** of normal random variables is also normally distributed itself.

Proofs are omitted.

6.2.2 Simulating bivariate normal random variables

R has built-in functions to simulate bivariate and multivariate normal random variables. However, we can simulate using the strategy introduced in section 6.1: first simulate from the marginal for X (or marginal for Y if preferred), and then simulate Y from the conditional $Y|X = x$.

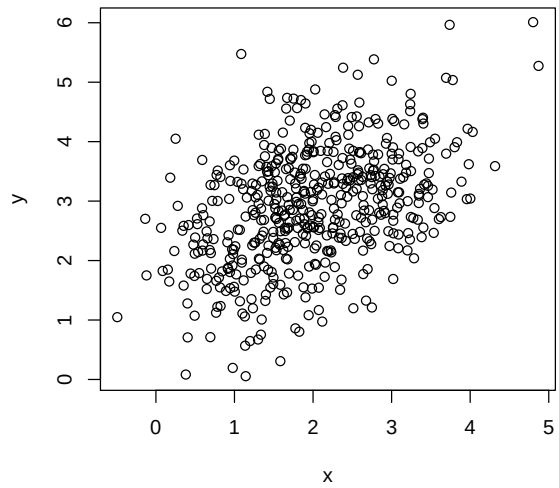
Example 6.3: Suppose $(X, Y)^T \sim N_2(\underline{\mu}, \Sigma)$ where $\mu_x = 2$, $\mu_y = 3$, $\sigma_x = 1$, $\sigma_y = 1$ and $\rho = 0.5$. Simulate a sample of size 500 from this distribution and draw a scatter plot. Use simulation to find $\Pr(X^2 + Y^2 < 9)$.

The marginal distribution of X is $X \sim N(2, 1^2)$. Using the formula for the conditional

$$\begin{aligned} Y|X = x &\sim N(\mu_y + \rho\frac{\sigma_y}{\sigma_x}(x - \mu_x), \sigma_y^2(1 - \rho^2)) \\ &\sim N(3 + 0.5(x - 2), 0.75). \end{aligned}$$

To sample:

```
npts = 500
x = rnorm(npts, mean = 2, sd = 1)
y = rnorm(npts, mean = 3+0.5*(x-2), sd = sqrt(0.75))
```



To find $\Pr(X^2 + Y^2 < 9)$ approximately, count the number of points in the region:

```
npts = 10000
x = rnorm(npts, mean = 2, sd = 1)
y = rnorm(npts, mean = 3+0.5*(x-2), sd = sqrt(0.75))
f = x^2+y^2
sum(f<9)/npts
[1] 0.2877
```

6.2.3 The multivariate normal distribution

The multivariate normal distribution is a distribution on vectors in $\mathbb{R}^n$. Suppose that $\underline{X}$ is a random vector with n entries, i.e. $\underline{X} = (X_1, \ldots, X_n)^T$. Then

$$\underline{X} \sim N_n(\underline{\mu}, \Sigma)$$

if $X_1, \ldots, X_n$ have joint PDF given by

$$f_{\underline{X}}(\underline{x}) = \frac{1}{2\pi\sqrt{\det \Sigma}} \exp\left(-\frac{1}{2}Q(\underline{x})\right)$$

where

$$Q(\underline{x}) = (\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu}).$$

This definition makes sense for any column vector $\underline{\mu} \in \mathbb{R}^n$ and any **positive definite** $n \times n$ matrix Σ. Some remarks:

1. The vector $\underline{\mu}$ is the **mean** of the distribution and Σ is called the **covariance** matrix.
2. All the marginal distributions of $\underline{X}$ are normal. (We do not specify their parameters here, however).
3. Similarly, all the conditional distributions of $\underline{X}$ are normal. (Again, we do not specify the parameters of these distributions here).

Example 6.4: Suppose $\underline{X} \sim N_3(0, \Sigma)$ where

$$\Sigma = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 3 \end{bmatrix}.$$

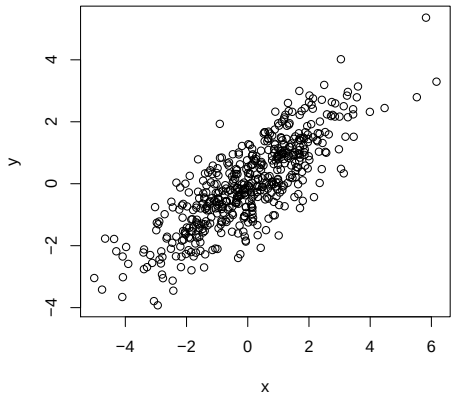
Use R to sample from this distribution. Create a scatterplot of (X, Y) samples and compute the correlation.

In R, you use the `mvrnorm` function to sample:

```
library(MASS)
npts = 500
mu = c(0, 0, 0)
sigma = matrix(data = c(3,2,1,2,2,1,1,1,3), nrow = 3, byrow = TRUE)
x = mvrnorm(npts, mu, sigma)
```

Now `x` has dimension `npts`$\times 3$, so column 1 is X and column 2 is Y:

```
plot(x[,1], x[,2], xlab = 'x', ylab = 'y')
```



The sample correlation is:

```
cor(x[,1],x[,2])
[1] 0.812629
```