

MAS2906: Computational Probability and Statistics with R

Newcastle University
`lee.fawcett@ncl.ac.uk`

Semester 1, 2021/22

Assessment changes

■ Numbas test 1 (R Refresher)

- Pushed back by one week, to allow an extended R refresher in class
- Will open Thursday 21 October 5pm
- Deadline: Friday 29th October, 4pm UK time
- Practice material released this coming Thursday, after the practical

★ Mini project

- New deadline: Friday 3rd December, 4pm UK time
- Will be released Thursday 11th November, 5pm

Assessment changes

■ Numbas test 1 (R Refresher)

- Pushed back by one week, to allow an extended R refresher in class
- Will open Thursday 21 October 5pm
- Deadline: **Friday 29th October, 4pm UK time**
- Practice material released this coming Thursday, after the practical

★ Mini project

- New deadline: **Friday 3rd December, 4pm UK time**
- Will be released Thursday 11th November, 5pm

Assessment changes

- Numbas test 1 (R Refresher)
 - Pushed back by one week, to allow an extended R refresher in class
 - Will open Thursday 21 October 5pm
 - Deadline: **Friday 29th October, 4pm UK time**
 - Practice material released this coming Thursday, after the practical
- Mini project
 - New deadline: Friday 3rd December, 4pm UK time
 - Will be released Thursday 11th November, 5pm

Assessment changes

- Numbas test 1 (R Refresher)
 - Pushed back by one week, to allow an extended R refresher in class
 - Will open Thursday 21 October 5pm
 - Deadline: **Friday 29th October, 4pm UK time**
 - Practice material released this coming Thursday, after the practical
- Mini project
 - New deadline: **Friday 3rd December, 4pm UK time**
 - Will be released Thursday 11th November, 5pm

This week's work

Main Aim: Recap everything studied in R over the last two weeks, with a focus on the hard bits: **functions**, **for loops** and **if statements** in R.

We will do this within the context of four simulation studies:

1. The sticker album: Covering Section 2.6 of the course notes (today)
2. Sampling distributions: Additional example (possibly today)
3. Monte Carlo integration: Online example (this week/early next week)
4. Monopoly: You'll love it! (Thursday's practical class)

This week's work

Main Aim: Recap everything studied in R over the last two weeks, with a focus on the hard bits: **functions**, **for loops** and **if statements** in R.

We will do this within the context of four **simulation studies**:

1. The sticker album: Covering Section 2.6 of the course notes (today)
2. Sampling distributions: Additional example (possibly today)
3. Monte Carlo integration: Online example (this week/early next week)
4. Monopoly: You'll love it! (Thursday's practical class)

2.6 Sticker album

We will now run through Section 2.6 of the course notes, thus completing our R Refresher – we will pull everything together through an interesting example!



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 the sticker album?”

2.6 Sticker album

Try out the following line of code a few times, and make sure you understand what's going on!

Initially, to simplify things, we'll assume there are only **50** stickers to collect.

```
# Let's buy a packet of stickers!

# Random sample of size 5 from {1,2,...,50}
(packet = sample(1:50, 5, replace=TRUE))
[1] 17 28 34  8  1

# We got stickers 1, 8, 17, 28 and 34!

(packet = sample(1:50, 5, replace=TRUE))
[1] 10 23 20 48 11

# Now we got stickers 10, 11, 20, 23 and 48!
```

2.6 Sticker album

Try out the following line of code a few times, and make sure you understand what's going on!

Initially, to simplify things, we'll assume there are only **50** stickers to collect.

```
1 # Let's buy a packet of stickers!
2
3 # Random sample of size 5 from {1, 2, ..., 50}
4 (packet = sample(1:50, 5, replace=TRUE))
5 [1] 17 28 34 8 1
6 # We got stickers 1, 8, 17, 28 and 34!
7
8 (packet = sample(1:50, 5, replace=TRUE))
9 [1] 10 23 20 48 11
10 # Now we got stickers 10, 11, 20, 23 and 48!
```

2.6 Sticker album

Now try out the next two blocks of code - again, make sure you follow what's going on.

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

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

2.6 Sticker album

The last block of code consisted of five lines, each one doing the same thing!

This is crying out for a for loop:

```
for(j in 1:5){  
  stickers[packet[j]] = 0}  
  
# Now look at what's happened to stickers (yours will be different!)  
stickers  
[1] 1 2 3 4 5 6 7 8 9 0 0 12 13 14 15 16  
[17] 17 18 19 0 21 22 0 24 25 26 27 28 29 30 31 32  
[33] 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 0  
[49] 49 50
```

2.6 Sticker album

The last block of code consisted of five lines, each one doing the same thing!

This is crying out for a **for loop**:

```
1 for(j in 1:5){
2     stickers[packet[j]] = 0}
3
4 # Now look at what's happened to stickers (yours will be different!)
5 stickers
6     [1]  1  2  3  4  5  6  7  8  9  0  0 12 13 14 15 16
7     [17] 17 18 19  0 21 22  0 24 25 26 27 28 29 30 31 32
8     [33] 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47  0
9     [49] 49 50
```

2.6 Sticker album

Now:

- Try buying another packet of stickers using the `sample` command
 - Run this packet of stickers through the `for` loop
 - Watch how the `stickers` vector is updated – you should see more zeros being added!
- 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...
- ... Once again, we are doing the same thing repeatedly, so another `for` loop is required!

2.6 Sticker album

Now:

- Try buying another packet of stickers using the `sample` command
- Run this packet of stickers through the `for` loop
- Watch how the `stickers` vector is updated – you should see more zeros being added!
- 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...

■ ... Once again, we are doing the same thing repeatedly, so another `for` loop is required!

2.6 Sticker album

Now:

- Try buying another packet of stickers using the `sample` command
- Run this packet of stickers through the `for` loop
- Watch how the `stickers` vector is updated – you should see more zeros being added!
- 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...
- ... Once again, we are doing the same thing repeatedly, so another **for loop** is required!

2.6 Sticker album

```
1 # Let's buy another packet of stickers!
2 # Random sample of size 5 from {1, 2, ..., 50}
3 (packet = sample(1:50, 5, replace=TRUE))
4 # And let's update the stickers sequence
5 for(j in 1:5){
6     stickers[packet[j]] = 0}
7
8 # And again
9 (packet = sample(1:50, 5, replace=TRUE))
10 for(j in 1:5){
11     stickers[packet[j]] = 0}
12
13 # And again!
14 (packet = sample(1:50, 5, replace=TRUE))
15 for(j in 1:5){
16     stickers[packet[j]] = 0}
```

Notice once again, we're doing something repeatedly... another for loop?!

2.6 Sticker album

```
1 # Re-set the sticker sequence
2 (stickers = seq(1, 50))
3 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
4 [17] 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
5 [33] 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
6 [49] 49 50
7
8 # Start a big for loop to repeatedly buy packets of stickers
9 for(i in 1:10000){ # 10,000 – something large!
10   packet = sample(1:50, 5, replace=TRUE)
11
12   for(j in 1:5){
13     stickers[packet[j]] = 0} # Close the j loop
14   } # Close the i loop
15
16 # Take a look at stickers:
17 stickers
18 [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
19 [26] 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
```

2.6 Sticker album

Looking at the listing above, it looks like we've filled the sticker album after buying 10,000 packets of stickers – 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 bought the 200th packet?
- Or the 1524th packet?!

We need to 'break' the code as soon as all entries in the sticker sequence are zero.

2.6 Sticker album

Looking at the listing above, it looks like we've filled the sticker album after buying 10,000 packets of stickers – 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 bought the 200th packet?
- ★ Or the 1524th packet?!

We need to 'break' the code as soon as all entries in the sticker sequence are zero.

2.6 Sticker album

Looking at the listing above, it looks like we've filled the sticker album after buying 10,000 packets of stickers – 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 bought the 200th packet?

■ Or the 1524th packet?!

We need to 'break' the code as soon as all entries in the sticker sequence are zero.

2.6 Sticker album

Looking at the listing above, it looks like we've filled the sticker album after buying 10,000 packets of stickers – 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 bought the 200th packet?

★ Or the 1524th packet?!

We need to 'break' the code as soon as all entries in the sticker sequence are zero.

2.6 Sticker album

Looking at the listing above, it looks like we've filled the sticker album after buying 10,000 packets of stickers – 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 bought the 200th packet?
- Or the 1524th packet?!

We need to 'break' the code as soon as all entries in the sticker sequence are zero.

2.6 Sticker album

Looking at the listing above, it looks like we've filled the sticker album after buying 10,000 packets of stickers – 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 bought the 200th packet?
- Or the 1524th packet?!

We need to 'break' the code as soon as all entries in the sticker sequence are zero.

2.6 Sticker album

Now edit your code, with the addition of a `break` command within an **if statement**:

```
1 stickers = seq(1, 50)
2
3 for(i in 1:10000){
4   packet = sample(1:50, 5, replace=TRUE)
5
6   for(j in 1:5){
7     stickers[packet[j]] = 0}
8
9 # Break the code when all entries in the sticker sequence are zero
10  if(sum(stickers)==0) {
11    break}
12  }
13
14 # When did the code break?
15 i
16 [1] 37
```

2.6 Sticker album

So this time it only took 37 packets of stickers to completely fill the album! (Yours may well be different)

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.

2.6 Sticker album

So this time it only took 37 packets of stickers to completely fill the album! (Yours may well be different)

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.

2.6 Sticker album

So this time it only took 37 packets of stickers to completely fill the album! (Yours may well be different)

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.

2.6 Sticker album

So this time it only took 37 packets of stickers to completely fill the album! (Yours may well be different)

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.

2.6 Sticker album

So this time it only took 37 packets of stickers to completely fill the album! (Yours may well be different)

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.

2.6 Sticker album

Now use the following code to make your FillAlbum function.

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

2.6 Sticker album

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

- The word `function` (line 2), 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 2)
- The use of braces surrounding the operations within the function (lines 2 and 14; all other braces relate to the for loops/if statement)
- A return statement (line 14), declaring what we would like the function to spit out

2.6 Sticker album

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

- The word `function` (line 2), 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 2)
- The use of braces surrounding the operations within the function (lines 2 and 14; all other braces relate to the for loops/if statement)
- A return statement (line 14), declaring what we would like the function to spit out

2.6 Sticker album

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

- The word `function` (line 2), 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 2)
- The use of `braces` surrounding the operations within the function (lines 2 and 14; all other braces relate to the for loops/if statement)
- A `return` statement (line 14), declaring what we would like the function to spit out

2.6 Sticker album

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

- The word `function` (line 2), 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 2)
- The use of `braces` surrounding the operations within the function (lines 2 and 14; all other braces relate to the for loops/if statement)
- A `return` statement (line 14), declaring what we would like the function to spit out

2.6 Sticker album

Now execute your `FillAlbum` function – first for 50 stickers, and then for the full 682!

```
# 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
```

2.6 Sticker album

Now execute your FillAlbum function – first for 50 stickers, and then for the full 682!

```
1 # Let's try for an album containing just 50 stickers again
2 FillAlbum(10000, 50)
3 [1] 52
4 FillAlbum(10000, 50)
5 [1] 32
6 FillAlbum(10000, 50)
7 [1] 40
8
9 # Now let's try for the full album of 682 stickers!
10 FillAlbum(10000, 682)
11 [1] 1290
12 FillAlbum(10000, 682)
13 [1] 1611
14 FillAlbum(10000, 682)
15 [1] 1252
```

2.6 Sticker album

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.

2.6 Sticker album

Referring to the code in the listing above, we have simulated three people trying to fill the full *Panini* sticker album.

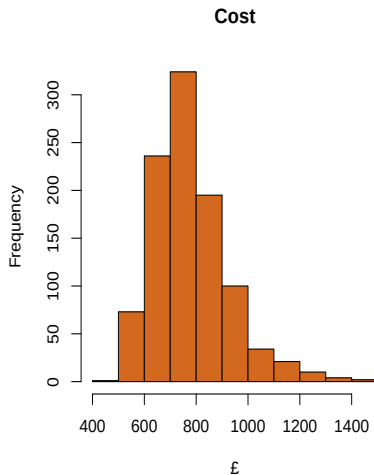
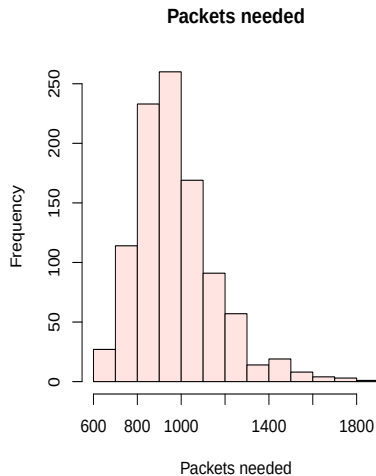
Respectively, they needed to buy 1,290, 1,611 and 1,252 packets of stickers to do this.

We could now execute this function many times, within another for loop.

2.6 Sticker album

```
1 # Let's simulate 1000 people trying to fill their albums
2 packets_needed = vector("numeric", 1000)
3
4 for(k in 1:1000){
5   packets_needed[k] = FillAlbum(10000, 682)}
6
7 cost = packets_needed*0.8      # Each packet costs 80p
8
9 mean(packets_needed); mean(cost)
10 [1] 974.609
11 [1] 779.6872
12
13 sd(packets_needed); sd(cost)
14 [1] 181.351
15 [1] 145.0808
16
17 par(mfrow=c(1,2))
18 hist(packets_needed, col="misty rose", xlab="Packets",
19      main="Packets needed")
19 hist(cost, col="chocolate", xlab="Pounds", main="Cost")
```

2.6 Sticker album



2.6 Sticker album

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)
- There is an associated cost of around £780 (standard deviation £145)
- The histograms 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!

2.6 Sticker album

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)
 - There is an associated cost of around £780 (standard deviation £145)
 - The histograms 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!

2.6 Sticker album

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)
- There is an associated cost of around £780 (standard deviation £145)
- The histograms 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!

2.6 Sticker album

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)
 - There is an associated cost of around £780 (standard deviation £145)
 - The histograms 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!

2.6 Sticker album

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)
- There is an associated cost of around £780 (standard deviation £145)
- The histograms 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!

Case study 2: Sampling distributions

Another case study: Sampling distributions

The results of this exercise support material we will consider towards the end of the semester, but the exercise itself lends itself well to checking your basic programming skills in R!

- (a) Commit the following code carefully to RStudio, and then produce a histogram of `x`. What can you say about the shape of the distribution you see?

```
1 x = rexp(10000, 2)
```

Do you know what the function `rexp` is actually doing?
(don't worry if not!) Can you hazard a guess?

Another case study: Sampling distributions

The results of this exercise support material we will consider towards the end of the semester, but the exercise itself lends itself well to checking your basic programming skills in R!

- (a) Commit the following code carefully to RStudio, and then produce a histogram of `x`. What can you say about the shape of the distribution you see?

```
1 x = rexp(10000, 2)
```

Do you know what the function `rexp` is actually doing?
(don't worry if not!) Can you hazard a guess?

Another case study: Sampling distributions

- (b) We now want to see what the distribution of the sample mean from x looks like.
- (i) Use the `sample` command to draw a random sample from x of length 30. Find the mean of this sample, and write it down.

(ii) Repeat what you did in (i) a further four times.

(iii) Now you should repeat what you did in part (i) 1000 times, but you should automate the process by using a `for` loop!

First of all, type the following code into RStudio, which will set up a vector of length 1000 to take sample means from repeated random samples from x :

```
xbar = vector("numeric", 1000)
```

Now at each iteration i in your `for` loop, $i = 1, \dots, 1000$, store the sample mean in position i of the vector `xbar`.

Another case study: Sampling distributions

- (b) We now want to see what the distribution of the sample mean from x looks like.
- (i) Use the `sample` command to draw a random sample from x of length 30. Find the mean of this sample, and write it down.
 - (ii) Repeat what you did in (i) a further four times.

(iii) Now you should repeat what you did in part (i) 1000 times, but you should automate the process by using a for loop! First of all, type the following code into RStudio, which will set up a vector of length 1000 to take sample means from repeated random samples from x :

```
xbar = vector("numeric", 1000)
```

Now at each iteration i in your for loop, $i = 1, \dots, 1000$, store the sample mean in position i of the vector `xbar`.

Another case study: Sampling distributions

- (b) We now want to see what the distribution of the sample mean from x looks like.
- (i) Use the `sample` command to draw a random sample from x of length 30. Find the mean of this sample, and write it down.
 - (ii) Repeat what you did in (i) a further four times.
 - (iii) Now you should repeat what you did in part (i) 1000 times...

but you should automate the process by using a `for` loop!

First of all, type the following code into RStudio, which will set up a vector of length 1000 to take sample means from repeated random samples from x :

```
xbar = vector("numeric", 1000)
```

Now at each iteration i in your `for` loop, $i = 1, \dots, 1000$, store the sample mean in position i of the vector `xbar`.

Another case study: Sampling distributions

- (b) We now want to see what the distribution of the sample mean from x looks like.
- (i) Use the `sample` command to draw a random sample from x of length 30. Find the mean of this sample, and write it down.
 - (ii) Repeat what you did in (i) a further four times.
 - (iii) Now you should repeat what you did in part (i) 1000 times... but you should automate the process by using a for loop!

First of all, type the following code into RStudio, which will set up a vector of length 1000 to take sample means from repeated random samples from x :

```
1 xbar = vector("numeric", 1000)
```

Now at each iteration i in your for loop, $i = 1, \dots, 1000$, store the sample mean in position i of the vector `xbar`.

Another case study: Sampling distributions

- (b) We now want to see what the distribution of the sample mean from x looks like.
- (i) Use the `sample` command to draw a random sample from x of length 30. Find the mean of this sample, and write it down.
 - (ii) Repeat what you did in (i) a further four times.
 - (iii) Now you should repeat what you did in part (i) 1000 times... but you should automate the process by using a for loop!

First of all, type the following code into RStudio, which will set up a vector of length 1000 to take sample means from repeated random samples from x :

```
1 xbar = vector("numeric", 1000)
```

Now at each iteration i in your for loop, $i = 1, \dots, 1000$, store the sample mean in position i of the vector `xbar`.

Another case study: Sampling distributions

- (c) The *Central Limit Theorem*, which we will come back to later in this module, states that – regardless of the distribution of the population – the distribution of the sample mean will be Normal, with mean μ and variance σ^2/n , where
- μ is the population mean
 - σ^2 is the population variance
 - n is the sample size (e.g. $n = 30$ in part (b))

Suppose your vector x contains the population. Can you verify the Central Limit Theorem for the distribution of your sample means obtained in (b)(iii)?

- (d) Write a function that explores the Central Limit Theorem for different size samples from x , where the sample size n is an argument to be passed in the function.

Another case study: Sampling distributions

- (c) The *Central Limit Theorem*, which we will come back to later in this module, states that – regardless of the distribution of the population – the distribution of the sample mean will be Normal, with mean μ and variance σ^2/n , where
- μ is the population mean
 - σ^2 is the population variance
 - n is the sample size (e.g. $n = 30$ in part (b))

Suppose your vector `x` contains the population. Can you verify the Central Limit Theorem for the distribution of your sample means obtained in (b)(iii)?

(d) Write a function that explores the Central Limit Theorem for different size samples from `x`, where the sample size n is an argument to be passed in the function.

Another case study: Sampling distributions

- (c) The *Central Limit Theorem*, which we will come back to later in this module, states that – regardless of the distribution of the population – the distribution of the sample mean will be Normal, with mean μ and variance σ^2/n , where
- μ is the population mean
 - σ^2 is the population variance
 - n is the sample size (e.g. $n = 30$ in part (b))

Suppose your vector `x` contains the population. Can you verify the Central Limit Theorem for the distribution of your sample means obtained in (b)(iii)?

- (d) Write a function that explores the Central Limit Theorem for different size samples from `x`, where the sample size n is an argument to be passed in the function.

Case study 3: Monte Carlo integration

Another case study: Monte Carlo integration

Suppose the random variable Z follows a **standard Normal distribution**, that is, $Z \sim N(0, 1)$.

Then Z has probability density function given by

$$f_Z(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}, \quad z \in \mathcal{R}.$$

Commit the following code to RStudio, and make sure you understand what's going on.

```
1 z = seq(-4, 4, 0.001)
2 d = 1/(sqrt(2*pi))*exp(-(z^2)/2)
3 plot(z, d, xlab = "z", ylab = "density", type = "l")
```

Another case study: Monte Carlo integration

Now suppose we want to find $\Pr(-2 < Z < 1)$.

The old-school way of doing this might be to annotate a diagram such as that produced on the previous slide, and then use **statistical tables**!

We're not going to do this. We're going to use *Monte Carlo integration*!

Another case study: Monte Carlo integration

Consider throwing random points at the plot you produced earlier, across a “simulation grid” that covers the area you’re interested in.

First, draw the “simulation grid”:

```
1 # Draw the simulation grid on your plot
2 # Lower bound for  $\Pr(-2 < Z < 1)$ 
3 abline(v = -2, lty = 2, col = "red")
4
5 # Upper bound for  $\Pr(-2 < Z < 1)$ 
6 abline(v = 1, lty = 2, col = "red")
7
8 # Covers the top of the Normal PDF
9 abline(h = 0.4, lty = 2, col = "red")
```

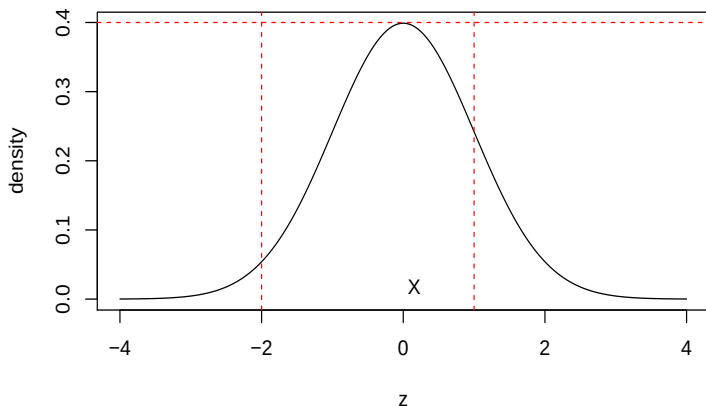
Another case study: Monte Carlo integration

Now throw random points at the grid, repeatedly – and watch how your plot updates.

```
1 # Generate a sequence for the x co-ordinate (between -2 and 1)
2 xseq = seq(-2, 1, 0.0001)
3
4 # Generate sequence for the y co-ordinate (between 0 and 0.4)
5 yseq = seq(0, 0.4, 0.0001)
```

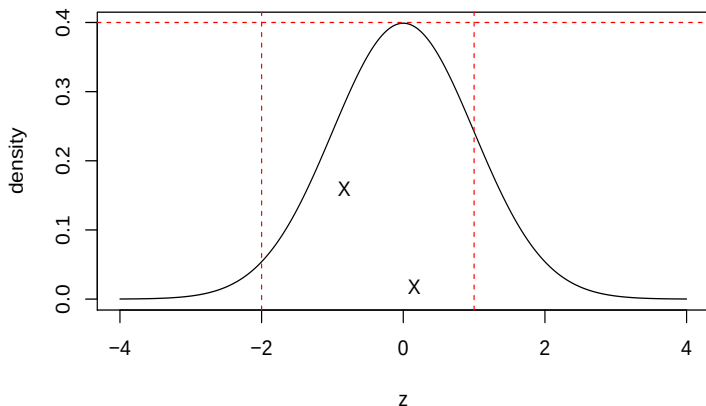
Another case study: Monte Carlo integration

```
1 # Throw your first point at the grid!  
2 lines(sample(xseq,1),sample(yseq,1),type="p",pch="X")
```



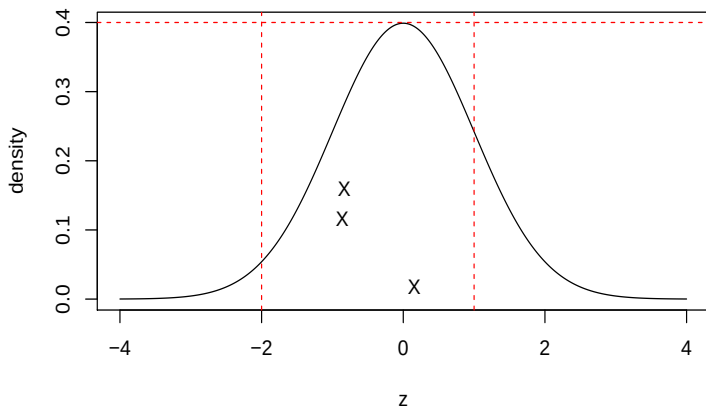
Another case study: Monte Carlo integration

```
1 # And another!  
2 lines(sample(xseq,1), sample(yseq,1), type="p", pch="X")
```



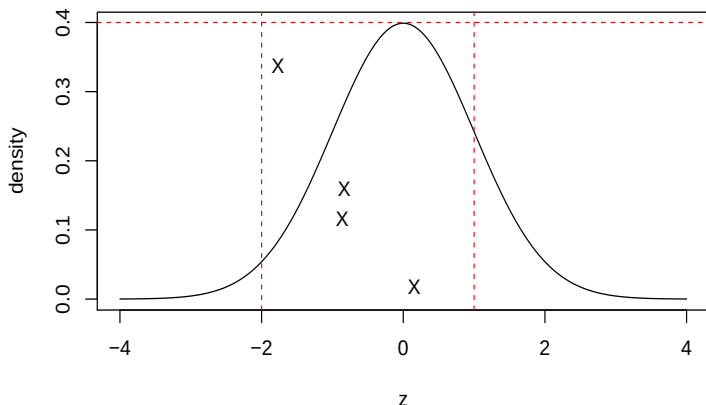
Another case study: Monte Carlo integration

```
1 # Keep going...  
2 lines(sample(xseq,1), sample(yseq,1), type="p", pch="X")
```



Another case study: Monte Carlo integration

```
1 # One more for now...!  
2 lines(sample(xseq,1), sample(yseq,1), type="p", pch="X")
```



Another case study: Monte Carlo integration

Observations

- As you throw random points at the grid — some will lie under the curve, some will lie above the curve
 - What's the probability that a point will lie under the curve (call this A)?

$$\Pr(A) = \frac{\text{Area under the curve}}{\text{Area of simulation grid}}$$

- Therefore, by rearranging, we can see that

$$\text{Area under the curve} = \Pr(A) \times \text{Area of simulation grid}$$

$$\Pr(-2 < Z < 1) \approx \frac{\# \text{points under the curve}}{\# \text{points simulated}} \times 1.2.$$

Another case study: Monte Carlo integration

Observations

- As you throw random points at the grid — some will lie under the curve, some will lie above the curve
- What's the probability that a point will lie under the curve (call this A)?

$$\Pr(A) = \frac{\text{Area under the curve}}{\text{Area of simulation grid}}$$

■ Therefore, by rearranging, we can see that

$$\text{Area under the curve} = \Pr(A) \times \text{Area of simulation grid}$$

$$\Pr(-2 < Z < 1) \approx \frac{\# \text{points under the curve}}{\# \text{points simulated}} \times 1.2.$$

Another case study: Monte Carlo integration

Observations

- As you throw random points at the grid — some will lie under the curve, some will lie above the curve
- What's the probability that a point will lie under the curve (call this A)?

$$\Pr(A) = \frac{\text{Area under the curve}}{\text{Area of simulation grid}}.$$

■ Therefore, by rearranging, we can see that

$$\text{Area under the curve} = \Pr(A) \times \text{Area of simulation grid}$$

$$\Pr(-2 < Z < 1) \approx \frac{\# \text{points under the curve}}{\# \text{points simulated}} \times 1.2.$$

Another case study: Monte Carlo integration

Observations

- As you throw random points at the grid — some will lie under the curve, some will lie above the curve
- What's the probability that a point will lie under the curve (call this A)?

$$\Pr(A) = \frac{\text{Area under the curve}}{\text{Area of simulation grid}}.$$

- Therefore, by rearranging, we can see that

$$\text{Area under the curve} = \Pr(A) \times \text{Area of simulation grid}$$

$$\Pr(-2 < Z < 1) \approx \frac{\text{points under the curve}}{\text{points simulated}} \times 1.2.$$

Another case study: Monte Carlo integration

Observations

- As you throw random points at the grid — some will lie under the curve, some will lie above the curve
- What's the probability that a point will lie under the curve (call this A)?

$$\Pr(A) = \frac{\text{Area under the curve}}{\text{Area of simulation grid}}.$$

- Therefore, by rearranging, we can see that

$$\text{Area under the curve} = \Pr(A) \times \text{Area of simulation grid}$$

$$\Pr(-2 < Z < 1) \approx \frac{\text{points under the curve}}{\text{points simulated}} \times 1.2.$$

Another case study: Monte Carlo integration

Observations

- As you throw random points at the grid — some will lie under the curve, some will lie above the curve
- What's the probability that a point will lie under the curve (call this A)?

$$\Pr(A) = \frac{\text{Area under the curve}}{\text{Area of simulation grid}}.$$

- Therefore, by rearranging, we can see that

$$\begin{aligned} \text{Area under the curve} &= \Pr(A) \times \text{Area of simulation grid} \\ \Pr(-2 < Z < 1) &\approx \frac{\text{\#points under the curve}}{\text{\#points simulated}} \times 1.2. \end{aligned}$$

Another case study: Monte Carlo integration

So, very crudely,

$$\Pr(-2 < Z < 1) \approx \frac{3}{4} \times 1.2 = 0.9,$$

according to the points on my plot a few slides ago.

- This approximation should improve as we throw more and more points at the grid
- We can do this easily – many times – by using a `for loop`
- We can check our answers using tables (or even using one of R's built-in functions)
- Generally, such **Monte Carlo integration** is really useful if our integration problem is too hard to do by hand

Another case study: Monte Carlo integration

So, very crudely,

$$\Pr(-2 < Z < 1) \approx \frac{3}{4} \times 1.2 = 0.9,$$

according to the points on my plot a few slides ago.

- This approximation should improve as we throw more and more points at the grid
- We can do this easily – many times – by using a `for` loop
- We can check our answers using tables (or even using one of R's built-in functions)
- Generally, such **Monte Carlo integration** is really useful if our integration problem is too hard to do by hand

Another case study: Monte Carlo integration

So, very crudely,

$$\Pr(-2 < Z < 1) \approx \frac{3}{4} \times 1.2 = 0.9,$$

according to the points on my plot a few slides ago.

- This approximation should improve as we throw more and more points at the grid
- We can do this easily – many times – by using a for loop
- We can check our answers using tables (or even using one of R's built-in functions)
- Generally, such Monte Carlo integration is really useful if our integration problem is too hard to do by hand

Another case study: Monte Carlo integration

So, very crudely,

$$\Pr(-2 < Z < 1) \approx \frac{3}{4} \times 1.2 = 0.9,$$

according to the points on my plot a few slides ago.

- This approximation should improve as we throw more and more points at the grid
- We can do this easily – many times – by using a **for loop**
- We can check our answers using tables (or even using one of R's built-in functions)
- Generally, such Monte Carlo integration is really useful if our integration problem is too hard to do by hand

Another case study: Monte Carlo integration

So, very crudely,

$$\Pr(-2 < Z < 1) \approx \frac{3}{4} \times 1.2 = 0.9,$$

according to the points on my plot a few slides ago.

- This approximation should improve as we throw more and more points at the grid
- We can do this easily – many times – by using a **for loop**
- We can check our answers using tables (or even using one of R's built-in functions)

★ Generally, such Monte Carlo integration is really useful if our integration problem is too hard to do by hand

Another case study: Monte Carlo integration

So, very crudely,

$$\Pr(-2 < Z < 1) \approx \frac{3}{4} \times 1.2 = 0.9,$$

according to the points on my plot a few slides ago.

- This approximation should improve as we throw more and more points at the grid
- We can do this easily – many times – by using a **for loop**
- We can check our answers using tables (or even using one of R's built-in functions)
- Generally, such **Monte Carlo integration** is really useful if our integration problem is too hard to do by hand

Another case study: Monte Carlo integration

```
1 MCIntegrate = function(N){
2   no_of_hits = 0
3   for(i in 1:N){
4     # Generate the x/y co-ordinates in a better way:
5     x = runif(1, -2, 1)
6     y = runif(1, 0, 0.4)
7
8     # Evaluate the Normal density at x:
9     d = 1/(sqrt(2*pi))*exp(-(x^2)/2)
10    if(y<d){      # Is the point under the curve?
11      no_of_hits = no_of_hits+1
12    }
13  }
14  P = no_of_hits/N
15  area_under_curve = P*(3*0.4)
16  return(area_under_curve)}
```

Find P

Find the area:

Another case study: Monte Carlo integration

- (a) Run the function `MCintegrate` using $N = 10, 100, 1000$ and $10,000$.
- (b) Can you generalise the function so that it can be used to find probabilities from the standard Normal distribution between *any* two points (as opposed to just between -2 and 1)?
- (c) Explore the R's built-in function `pnorm` to evaluate the integral exactly, and compare to your answers in part (a).