Chance Processes and Variability

Intro to Stats, Spring 2017

Prof. Gaston Sanchez

Learning Objectives

- How to use R to simulate chance processes
- Getting to know the function `sample()`
- Simulate flipping a coin
- Simulate rolling a die
- Simulate drawing tickets from a box

Introduction

In this tutorial we will see how to use R to simulate basic chance processes like tossing a coin, rolling a die, or drawing tickets from a box. The aim is to give you some tools that allow you to better understanding and visualize fundamental concepts such as the law of large numbers, the law of averages, and the central limit theorem.

Coins, Dice, and Boxes with Tickets

Chance processes, also referred to as chance experiments, have to do with actions in which the resulting outcome turns out to be different in each occurrence.

Typical examples of basic chance processes are tossing one or more coins, rolling one or more dice, selecting one or more cards from a deck of cards, and in general, things that can be framed in terms of drawing tickets out of a box (or any other type of container: bag, urn, etc.).

You can use your computer, and R in particular, to simulate chances processes. In order to do that, the first step consists of learning how to create a virtual coin, or die, or box with tickets.

Creating a coin

The simplest way to create a coin with two sides, `"heads"` and `"tails"`, is with an R vector via the *combine* function `c()`

```
coin = c("heads", "tails")
```

You can also create a *numeric* coin that shows `0` and `1` instead of `"heads"` and `"tails"`:

```
coin = c(0, 1)
```

Creating a die

What about simulating a die in R? Pretty much the same way you create a coin: simply define a vector with numbers representing the number of spots in a die.

```
die = c(1, 2, 3, 4, 5, 6)

# equivalent
die = 1:6
```

Creating a box with tickets

Likewise, you can create a general box with tickets. For instance, say you have a box with tickets labeled 1, 2, 3 and 4; this can be implemented in R as:

```
tickets = c(1, 2, 3, 4)
```

Drawing tickets with `sample()`

Once you have an object that represents the *box with tickets*, the next step involves learning how to draw tickets from the box. One way to simulate drawing tickets from a box in R is with the function `sample()` which lets you draw random samples, with or without replacement, from an input vector.

For example, consider a "box" with tickets 1, 2, 3. To draw one ticket, use `sample()` like this:

```
# box with tickets
tickets = c(1, 2, 3)

# draw one ticket
sample(tickets, size = 1)
```

```
## [1] 1
```

By default, `sample()` draws each ticket with the same probability. In other words, ecah ticket is assigned the same probability of being chosen. Another default behavior of `sample()` is to take a sample of the specified `size` without replacement. If `size = 1`, it does not really matter whether the sample is done with or without replacement.

To draw two tickets WITHOUT replacement, use `sample()` like this:

```
# draw 2 tickets without replacement
sample(tickets, size = 2)
```

```
## [1] 1 3
```

To draw two tickets WITH replacement, use `sample()` and specify its argument `replace = TRUE`, like this:

```r
# draw 2 tickets with replacement
sample(tickets, size = 2, replace = TRUE)
```

```
## [1] 1 3
```

The way `sample()` works is by taking a random sample from the input vector. This means that every time you invoke `sample()` you will likely get a different output.

In order to make the examples replicable (so you can get the same output as me), you need to specify what is called a **random seed**. This is done with the function `set.seed()`. By setting a *seed*, every time you use one of the random generator functions, like `sample()`, you will get the same values.

```r
# set random seed
set.seed(1257)

# draw 4 tickets with replacement
sample(tickets, size = 4, replace = TRUE)
```

```
## [1] 1 3 2 1
```

Try the code above. You should get the exact same sample.

Last but not least, `sample()` comes with the argument `prob` which allows you to provide specific probabilities for each element in the input vector.

By default, `prob = NULL`, which means that every element has the same probability of being drawn. In the example of tossing a coin, the command `sample(coin)` is equivalent to `sample(coin, prob = c(0.5, 0.5))`. In the latter case we explicitly specify a probability of 50% chance of heads, and 50% chance of tails:

```
## [1] "tails" "heads"
```

```
## [1] "tails" "heads"
```

However, you can provide different probabilities for each of the elements in the input vector. For instance, to simulate a **loaded** coin with chance of heads 20%, and chance of tails 80%, set `prob = c(0.2, 0.8)` like so:

```r
# tossing a loaded coin (20% heads, 80% tails)
sample(coin, size = 5, replace = TRUE, prob = c(0.2, 0.8))
```

```
## [1] "heads" "tails" "tails" "tails" "tails"
```

Simulating tossing a coin

Now that we've talked about `sample()`, let's use R to implement code that simulates tossing a fair coin one or more times.

Recap. To toss a coin using R, we first need an object that plays the role of a coin. A simple way to create a `coin` is using a vector with two elements: `"heads"` and `"tails"`. Then, to simulate tossing a coin one or more times, we use the `sample()` function. Here's how to simulate a coin toss using `sample()` to take a random sample of size 1 from `coin`:

```
# coin object
coin <- c("heads", "tails")

# one toss
sample(coin, size = 1)
```

```
## [1] "tails"
```

To simulate multiple tosses, just change the `size` argument, and specify sampling with replacement (`replace = TRUE`):

```
# 3 tosses
sample(coin, size = 3, replace = TRUE)
```

```
## [1] "tails" "heads" "tails"
```

```
# 6 tosses
sample(coin, size = 6, replace = TRUE)
```

```
## [1] "heads" "heads" "heads" "tails" "heads" "tails"
```

Coin Simulations

Now that we have all the elements to toss a coin with R, let's simulate flipping a coin 100 times, and use the function `table()` to count the resulting number of `"heads"` and `"tails"`:

```
# number of flips
num_flips = 100

# flips simulation
coin = c('heads', 'tails')
flips = sample(coin, size = num_flips, replace = TRUE)

# number of heads and tails
freqs = table(flips)
freqs
```

```
## flips
## heads tails
##    57    43
```

In my case, I got 57 heads and 43 tails. Your results will probably be different than mine. Some of you will get more `"heads"`, some of you will get more `"tails"`, and some will get exactly 50 `"heads"` and 50 `"tails"`.

Run another series of 100 flips, and find the frequency of `"heads"` and `"tails"`:

```
flips = sample(coin, size = num_flips, replace = TRUE)
freqs = table(flips)
freqs
```

```
## flips
## heads tails
##    47    53
```

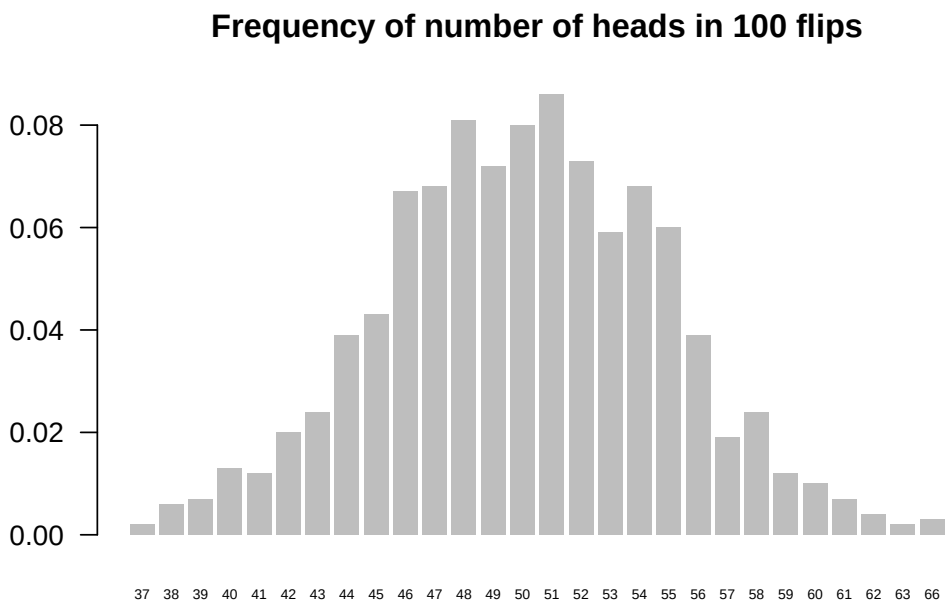
Let's make things a little bit more complex but also more interesting. The idea is to repeat 100 flips 1000 times. To carry out this simulation, we are going to use a programming structure called a `for` loop. This is one way to tell the computer to repeat the same action a given number of times. Don't worry about this. Just execute the following lines of code:

```
# total number of repetitions
times = 1000

# "empty" vectors to store number of heads and tails in each repetition
heads = c(0, times)
tails = c(0, times)

# 100 flips of a coin, repeated 1000 times
for (i in 1:times) {
  flips = sample(coin, size = 100, replace = TRUE)
  freqs = table(flips)
  heads[i] = freqs[1]
  tails[i] = freqs[2]
}
```

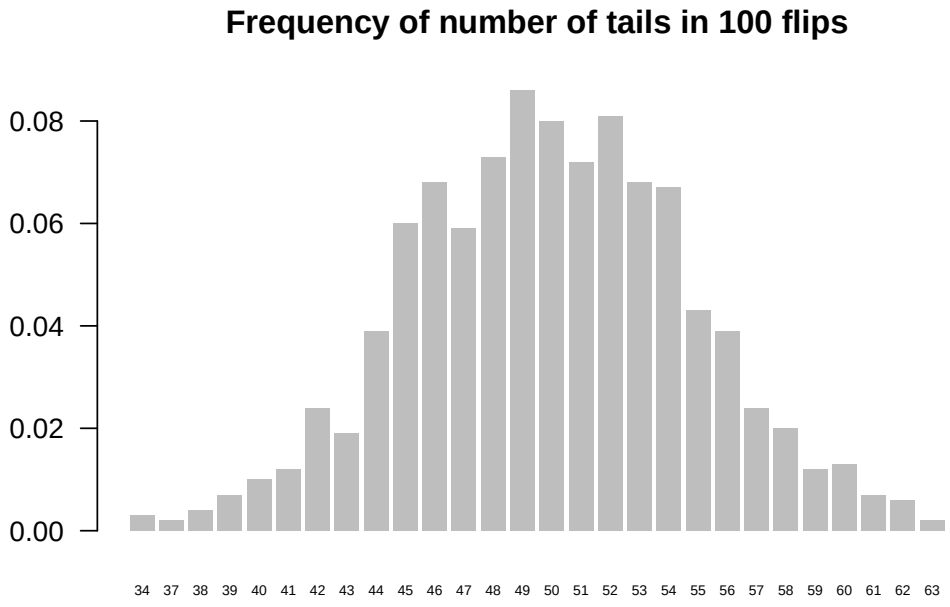
What the code above is doing is simulating 100 flips of a coin, not once, not twice, but 1000 times. In each repetition, we count how many `"heads"` and how many `"tails"`, and store those counts in the vectors `heads` and `tails`, respectively.

Each vector, `heads` and `tails`, contains 1000 values. Moreover, we can get a histogram to see the empirical relative frequency:

```
barplot(table(heads)/1000, las = 1, cex.names = 0.5, border = NA,
        main = "Frequency of number of heads in 100 flips")
```




```
barplot(table(tails)/1000, las = 1, cex.names = 0.5, border = NA,
        main = "Frequency of number of tails in 100 flips")
```



Frequencies

Typical probability problems that have to do with coin tossing, require to compute the total proportion of `"heads"` and `"tails"`:

```
# five tosses
five <- sample(coin, size = 5, replace = TRUE)
```

```r
# proportion of heads
sum(five == "heads") / 5
```

```
## [1] 0.6
```

```r
# proportion of tails
sum(five == "tails") / 5
```

```
## [1] 0.4
```

It is also customary to compute the relative frequencies of `"heads"` and `"tails"` in a series of tosses:

```r
# relative frequencies of heads
cumsum(five == "heads") / 1:length(five)
```

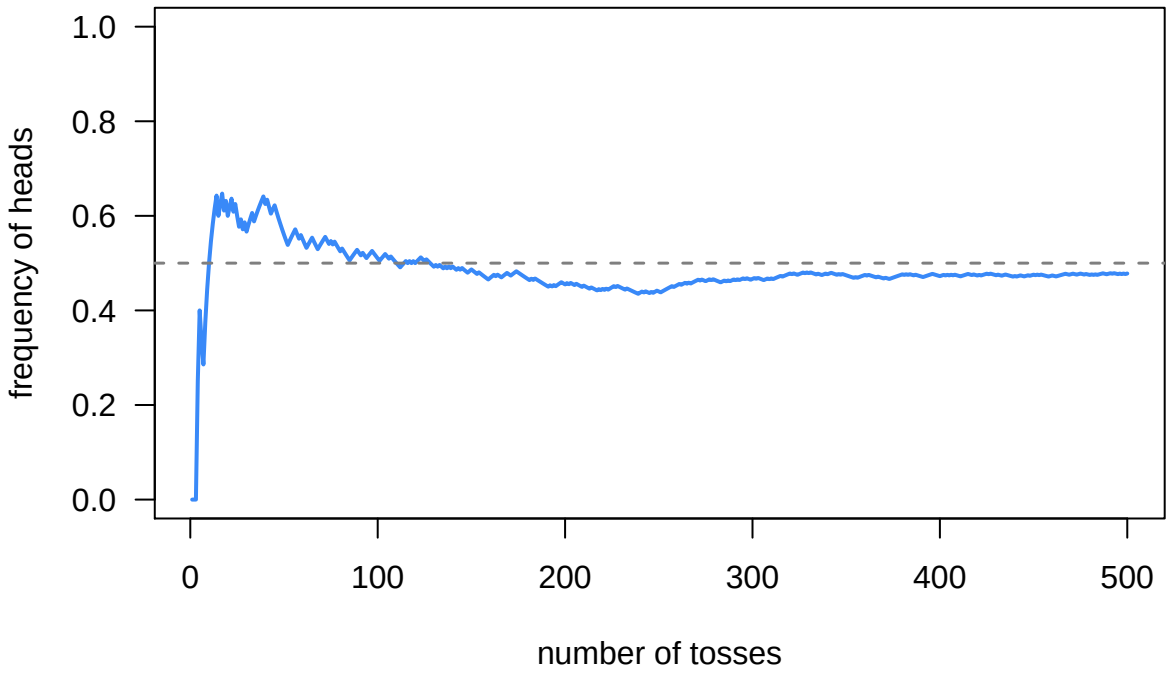
```
## [1] 1.0000000 0.5000000 0.6666667 0.7500000 0.6000000
```

```r
# relative frequencies of tails
cumsum(five == "tails") / 1:length(five)
```

```
## [1] 0.0000000 0.5000000 0.3333333 0.2500000 0.4000000
```

Likewise, it is common to look at how the relative frequencies of heads or tails change over a series of tosses:

```r
set.seed(5938)
hundreds <- sample(coin, size = 500, replace = TRUE)
head_freqs = cumsum(hundreds == "heads") / 1:500

plot(1:500, head_freqs, type = "l", ylim = c(0, 1), las = 1,
     col = "#3989f8", lwd = 2,
     xlab = 'number of tosses',
     ylab = 'frequency of heads')
# reference line at 0.5
abline(h = 0.5, col = 'gray50', lwd = 1.5, lty = 2)
```



So far we have written code in R that simulates tossing a coin one or more times. We have included commands to compute proportion of heads and tails, as well the relative frequencies of heads (or tails) in a series of tosses. In addition, we have produced a plot of the relative frequencies and see how, as the number of tosses increases, the frequency of heads (and tails) approach 0.5.

Simulating rolling a die

Now that you know how to simulate flipping a coin one or more times, you can do the same to simulate rolling a die:

```
die = 1:6

# rolling a die once
sample(die, size = 1)
```

```
## [1] 5
```

```
# rolling a pair of dice
sample(die, size = 2, replace = TRUE)
```

```
## [1] 2 6
```

```
# rolling a die 5 times
sample(die, size = 5, replace = TRUE)
```

```
## [1] 1 5 1 4 4
```