Problem sheet 2

Quantitative Methods

Preparation: Have a look at chapter 19 (on functions) of Wickham and Grolemund's *R for Data Science* and this brief tutorial on lists in R. Then, read the following:

A *loop* is a programming technique that allows us to repeat a set of operations multiple times in a compact manner. In R, the `for()` loop takes the following syntax:

```
for (i in X) {
  # Operation 1
  # Operation 2
  ...
  # Operation N
}
```

Here, `i` (or any other letter or name you want to choose) is a loop counter that controls the iterations of the loop, and `X` is the vector of values (of dimension `N`) that the loop counter will successively take on. Each operation is repeated for each value of `i`. Say we want to list the first five letters of the alphabet, using R's built-in vector called `letters`:

```
for (i in letters[1:5]) {
  print(i) # Print the i'th letter
}

## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
## [1] "e"
```

The `while()` loop takes the following syntax:

```
while (condition) {
  # Execute operation as long as condition == TRUE
}
```

The loop contents will be executed repeatedly so long as the logical value of `condition` is `TRUE`. Say we want to enumerate the elements of a sequence of natural numbers as long as their value is less than 7:

```r
x <- 1 # Set initial value
while (x < 7) { # Set limit to sequence
  print(x) # Output
  x <- x + 1 # Move to next number in sequence
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

The `sample()` function takes a random sample of a chosen size from the elements of vector, with or without replacement. For instance, if we want to take a random sample of three numbers between 1 and 10 with equal probability, we could do the following:

```r
sample(1:10, size = 3, replace = TRUE, prob = rep(0.1, 10))
```

```
## [1]  9  6 10
```

Try to type in all of the above commands yourself and make sure you understand what each step does.

A. Suppose you play a betting game where each bet is worth £1. You start with £S and continue to play until you either have £K or you owe £K. For each bet, the probability of winning is 0.49 and the probability of losing is 0.51. Such a process is known as a *random walk*.

1. Using the `while()` and `sample()` commands, create a function called `walk()` that simulates this betting game. Let `walk()` have two arguments: the starting value S and the boundary condition K. Set the default values of S and K to 0 and 10, respectively. Let the output of the function be the sequence of outcomes of each bet (i.e., a vector recording the money you have after each bet in the game).

2. Test your function to make sure it works. (Use `set.seet(281020)` so that your outputs are replicable. Note that this number happens to be the date when this problem set is due, but you can use any number to set the seed.)

3. Use the `replicate()` function to play ten rounds of the game. You will now have a list with ten entries, one for each round. Convert your list to a data frame (or tibble) named `walks`. (Hint: first convert each entry in your list to a data frame via a loop, then bind all those data frames together using e.g. `bind_rows()`.)

4. Generate a plot that visualises each of these ten rounds, with the total number of bets on the X-axis and the outcome of each bet on the Y-axis.

5. By playing 100,000 rounds, calculate the probability of winning the game. (Hint: use a `for()` loop and save the number of wins across all rounds.)

6. What is the probability of winning a game given that you have lost the first three bets?

B.[1] The Enigma machine is a famous encryption device, most notoriously used during World War II for military communication. In this exercise, we will analyse a simplified version of the Enigma machine which has 5 rotors, each of which comes with 10 pins with numbers ranging from 0 to 9. The machine also has a plugboard containing 26 holes, one for each letter of the alphabet, with 13 cables connecting all possible pairs of letters. (Look at the relevant Wikipedia entry to get a better sense of what the machine looks like.)

To either encode or decode a message, the machine must be fed a correct 5-digit passcode to align the rotors (much like a combination lock) and the plugboard must be correctly configured. A sender types a message on the keyboard and the plugboard scrambles the letters according to some algorithmic rule before the message is sent off in encrypted form. A receiver decodes this message by re-typing it on a paired Enigma machine that has the same passcode and plugboard configuration.

1. How many different 5-digit passcodes can be set on the 5 rotors?

2. How many possible configurations does the plugboard provide? In other words, in how many ways can 26 letters be divided into 13 pairs? (Make sure you do not count any pair twice: think of the example in the first lecture of how many ways there are to divide a group of four people into two groups of two.)

3. Based on the above, what is the total number of possible settings for the Enigma machine?

4. Three machines have been developed to decode 1,000 messaged encrypted by Enigma. We have the following information about each of these machines:

   - ⋆ Machine A is assigned 300 messages and has a failure rate of 9% (i.e., it fails to decode 9% of all messages it receives).
   - ⋆ Machine B is assigned 500 messages and has a failure rate of 4%.
   - ⋆ Machine C is assigned 200 messages and has a failure rate of 11%.

   We select a message at random from the pool of all 1,000 messages and find that this message has not been correctly decoded. Which machine is most likely responsible for this misake?

[1] This exercise draws on Exercise 6.6.1 in Imai's *Quantitative Social Science: An Introduction*, Chapter 6.

5. Create a 13×2 matrix representing a random configuration of the plugboard, showing pairs of letters (i.e., how one letter gets mapped to another).

6. Write a function named `enigma()` that encodes and decodes a message using the plugboard configuration in the previous question. The inputs of the function should be the message to be encoded or decoded and the plugboard configuration. The output should be the encoded or decoded message. (Hint: converting between upper- and lowercase letters by using `toupper()` or `tolower()` can help indicate which letters of a message need to be switched. To do the actual switching, the `gsub()` function might come in handy.)

Deadline: Submit a tidy and annotated R script via email by 2PM on Wednesday 28 October.