By the end of the practical you should feel confident writing and calling functions, and using if(), for() and while() constructions.

You should complete and understand questions 1–6 for next time.

1. Review

(a) Let \( t = 2 \): create a vector with \((i + 1)\)th entry \( e^{-t^{i+1}} \) for \( i = 0, \ldots, 10 \) (you might want to use the function factorial() for this).

(b) Write a function with arguments \( t \) and \( n \) that evaluates \( \sum_{i=0}^{n} e^{-t^{i+1}} \).

(c) Write your function again using a for() loop. Do not use vectors, or the sum() function. Check it gives the same answers as (b).

2. Solving a Quadratic. Write a function with three arguments \( a, b \) and \( c \), that returns the real roots of the equation \( ax^2 + bx + c = 0 \), if any. Your function should behave well if \( a = 0 \) and return an empty vector when there are no real roots.

3. Sieve of Eratosthenes. The Sieve of Eratosthenes is a method for finding all the prime numbers less than some specified \( n \). Here is an outline of the algorithm:

- Create a vector \( x \) of integers from 2 to \( n \), and an empty vector \( p \).
- Given \( x \), append the first element (say \( z \)) to \( p \); then remove any multiples of \( z \) (including \( z \) itself) from \( x \).
- Stop when \( x \) is empty, and return \( p \).

Write a function to implement this of Eratosthenes. It should take one argument \( n \), and return all the primes up to \( n \).

Try it for \( n = 10^3, 10^4, 10^5 \). [Optional: can you see any way to speed this procedure up?]

4. Random Walks. Write a function \texttt{rndwlk}, with an argument \( k \), that simulates a symmetric random walk (see lecture), stopping when the walk reaches \( k \) (or \(-k\)). After stopping it should return the entire walk.

Try calling \texttt{plot(rndwlk(10))} a few times to see how it looks.

5. Simulating Discrete Distributions. In lectures you’ve seen that we can sample \( X \) from a discrete distribution on \( \{1, \ldots, k\} \) as follows: let \( p_i = P(X = i) \). Then:

- generate \( U \sim \text{Unif}[0,1] \);
- set \( X = \min\{i \mid \sum_{j=1}^{i} p_j \geq U\} \).

Write a function that, given \( p \) containing \((p_1, \ldots, p_k)\) can simulate \( X \) from this distribution. You may find the functions \texttt{cumsum()} and \texttt{which()} useful.

Modify your function so that it takes an argument \( n \), and produces a vector of \( n \) i.i.d. values from the distribution \( p \). Comment on how you could check that your function worked as expected.
6. **Rejection Sampling.** We will write an R function to simulate $X \sim N(0, 1)$ using rejection sampling with the double exponential proposal. That is, from a random variable $Y$ with density

$$f_Y(y) = \exp(-|y|), \quad y \in \mathbb{R}.$$ 

(i) Write a function to simulate $n$ i.i.d. values of $Y$. [*Hint: you might want to start thinking about how to simulate an exponential random variable.]*

(ii) Write a function implementing rejection for $X$. The algorithm is:

1. simulate $Y \sim \exp(-|x|)$ and $U \sim U(0,1)$;
2. if $U < \exp(-Y^2/2 + |Y| - 1/2)$ accept $X = Y$ and stop. Otherwise repeat 1.

[*Hint: you can do this using a while statement. You should call the function you wrote in (a) to simulate $Y$. Your function should have no inputs, and return the simulated value of $X$.]*

(iii) Test your rejection sampler by simulating 1000 samples and checking they are normal using the `qqnorm()` function.

7. **Double for() Loop.** Using two `for()` loops, write a function with an argument $n$, which constructs the $n \times n$ matrix with entries $a_{ij} = i - j$.

8. **Moving Averages**

(a) Write a function to calculate the moving averages of length 3 of a vector $(x_1, \ldots, x_n)^T$. That is, it should return a vector $(z_1, \ldots, z_{n-2})^T$, where

$$z_i = \frac{1}{3} (x_i + x_{i+1} + x_{i+2}), \quad i = 1, \ldots, n-2.$$ 

Call this function `ma3()`.

(b) Write a function which takes two arguments, $x$ and $k$, and calculates the moving average of $x$ of length $k$. [Use a `for()` loop.]

(c) How does your function behave if $k$ is larger than (or equal to) the length of $x$? You can tell it to return an error in this case by using the `stop()` function. Do so.

(d) How does your function behave if $k = 1$? What should it do? Fix it if necessary.