Homework 4

Important: To submit your homework, please create your GitHub repository on the AU-R-Programming organization by Monday (Oct. 19th 2020) at 2pm and use this repository to work on this assignment. Your final submission will be done via Canvas in a single html file (in which you will specify the name of the corresponding GitHub repository) and is due by Friday, Oct. 30th 2020 at 11.59pm (no late work is accepted). The version submitted on Canvas will have to correspond to the last version on the GitHub repository.

To start, create a **private** GitHub repository and start its name with **HW4**. This project **must** be done using GitHub and respect the following requirements:

(1) All commit messages must be reasonably `clear` and `meaningful`.

(2) Include the code in the output necessary to replicate your results.

As always, you are free to consult online resources but you are not allowed to copy code. I have a control list of the main (R-related) resources available for the type of problems included in this assignment.

Problem 1: Royal Blood

Write a program that prints the numbers from 1 to 1000 with the following exceptions:

1. for multiples of three print “Royal” (instead of the number)
2. for multiples of five print “Blood” (instead of the number)
3. for numbers which are multiples of both three and five print “Royal Blood” (instead of the number).

An example output would be:

```
1, 2, Royal, 4, Blood, Royal, 7, 8, Royal, Blood, 11, Royal, 13, 14, Royal Blood, 16, 17
, Royal, 19, Blood, Royal, 22, 23, Royal, Blood, 26, Royal, 28, 29, Royal Blood, 31, 32,
Royal, 34, Blood, Royal, ...
```

Hope you can Figure It Out (https://youtu.be/GHX3lWJxQPA)! 😉

Problem 2: Let’s get things in order

For this problem you will have to use control structures to implement an algorithm which, applying it to a vector of integers of length n (let’s call it $\mathbf{a}$), works as follows:

1. Compare elements 1 and 2 of $\mathbf{a}$ and swap them if element 1 is larger than 2 (keep positions if 2 is larger or equal to 1)
2. Compare elements 2 and 3 of $\mathbf{a}$ and swap them if element 2 is larger than 3 (keep positions if 3 is larger or equal to 2)
3. Continue as per the previous points (comparing adjacent pairs of elements advancing along the vector) until you have compared (and eventually swapped) elements $n-1$ and n in $\mathbf{a}$
4. Repeat points 1 to 3 until you don’t have to swap any longer

Problem 3: To the bare minimum

In this problem we want to minimize a function (unimodal and convex), meaning that we would like to find the value of the argument (say x) which makes the function $f(x)$ as small as possible. To do so we will consider the function given by

$$f(x) = x^2 - 2x + 1.$$

To implement this function in R you can use the following code (execute it in your console):

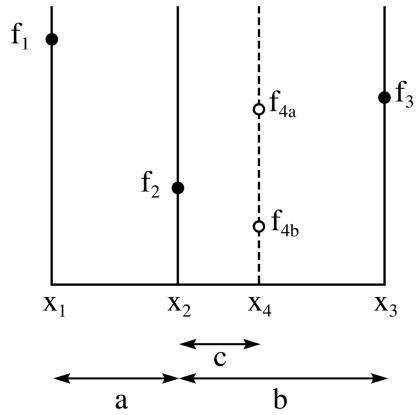
```
func <- function(x) x^2 - 2*x + 1
```

To get its value when $x = 2$, for example, then you can simply input

```
func(2)
```

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

To find its minimum we will make use of an algorithm that starts from two points within which we are sure the minimum is present. Let's call these points f_1 and f_3 which, in our case, will correspond to values of the function evaluated in the points $x_1 = -1$ and $x_3 = 3$. Using a seed, we then randomly select a value for x_2 such that $x_1 < x_2 < x_3$ and evaluate the function in this point (and call this value f_2). Given that the function is strictly convex, we have that f_2 is strictly smaller than f_1 and f_3. The diagram below illustrates this and will be used as a reference to illustrate the algorithm.



1. Define the current triplet of points as $t^{(i)}$: for example when we start the algorithm we have $i = 1$ and therefore the first triplet is $t^{(1)} = (x_1, x_2, x_3)$. We now evaluate the function in a fourth point x_4 which we will randomly pick within the largest range between the couple x_1 and x_2 on one side (the distance a in the above diagram) and the couple x_2 and x_3 on the other side (the distance b in the above diagram). Given that range b is largest, using a seed, we randomly pick a point x_4 such that $x_2 < x_4 < x_3$. In this point, we can have either of the following options:

    a. The function in point x_4 has value f_{4a}: our new triplet of points for the next step of the algorithm becomes $t^{(i+1)} = (x_1, x_2, x_4)$ since f_{4a} is larger than one of the external points in the current triplet $t^{(i)}$ (i.e. $f_{4a} > f_2$).
    b. The function in point x_4 has value f_{4b}: our new triplet of points for the next step of the algorithm becomes $t^{(i+1)} = (x_2, x_4, x_3)$ since f_{4b} is smaller than both the external points in the current triplet $t^{(i)}$ (i.e. $f_{4b} < f_2$ and $f_{4b} < f_3$).

2. Compare the minimum function values in the triplets $t^{(i)}$ and $t^{(i+1)}$ (i.e. compare the function in the central points of each triplet). Denoting the minimum of triplet $t^{(i)}$ as $f^{(i)}$ and the minimum of triplet $t^{(i+1)}$ as $f^{(i+1)}$ and $\delta = 10^{-5}$, compute $\Delta = |f^{(i)} - f^{(i+1)}|$.

    a. If $\Delta > \delta$ or if $\Delta = 0$, repeat from point 1 using the new triplet $t^{(i+1)}$ instead of $t^{(i)}$.
    b. If $\Delta \leq \delta$, stop the algorithm and output the central point in $t^{(i+1)}$.

Notice that if we are in situation (2.a.), then the triplets update as the algorithm goes on. For example, after the first run, $t^{(1)}$ gets replaced by $t^{(2)}$ in point 1, and $t^{(2)}$ is replaced by $t^{(3)}$ in points (1.a.) and (1.b.), and so on until $\Delta \leq \delta$.

Bonus points: An extra 5 bonus points will be given if you compare your solution to the true minimum and provide arguments to why this algorithm may not get close to the correct solution.

Bonus Problem (10 points): Application of Random Walk to Biology - Bacteria Motility

In this problem you will simulate bacteria searching for food in a 2D plane. Let us consider the problem where we observe $N = 20$ bacteria that are randomly moving on a 2D space and looking for food. One way for bacteria to move on such a space is the so-called “run and tumble” motion. More specifically, the bacterium swims in a given direction (running) and randomly changes its direction (tumbling). You can observe such a biological process in this video (https://www.youtube.com/watch?v=CATidoe0fDo&feature=youtu.be). This method can be modeled as a 2 dimensional random walk where the length of step S_t in a given random direction at time t is a random variable generated by $S_t \sim U(a, b)$ (where U represents the uniform distribution). For this exercise, we use $a = 0$ and $b = 2$.

We define the position of a bacterium at time t as $\begin{bmatrix} x_t \\ y_t \end{bmatrix}$. Then the position at the next step can be obtained by

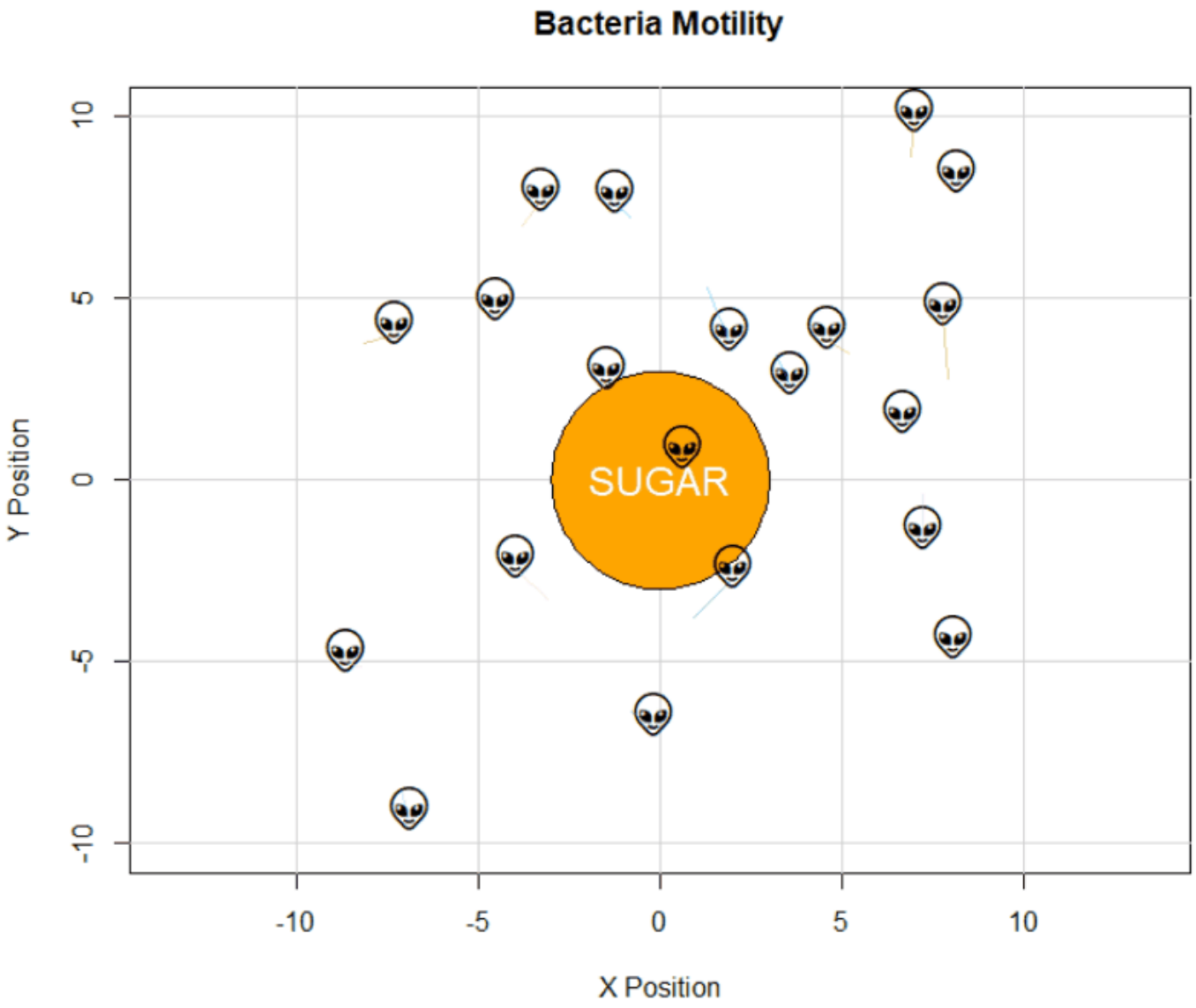
$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + S_t \begin{bmatrix} \cos(\delta_t) \\ \sin(\delta_t) \end{bmatrix},$$

where δ_t is the angle in radians defined as the random variable $\delta_t \sim U(0, 2\pi)$. For this exercise, we will consider the initial position of the i^{th} bacterium as $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$, where $x_0, y_0 \sim U(-9, 9)$.

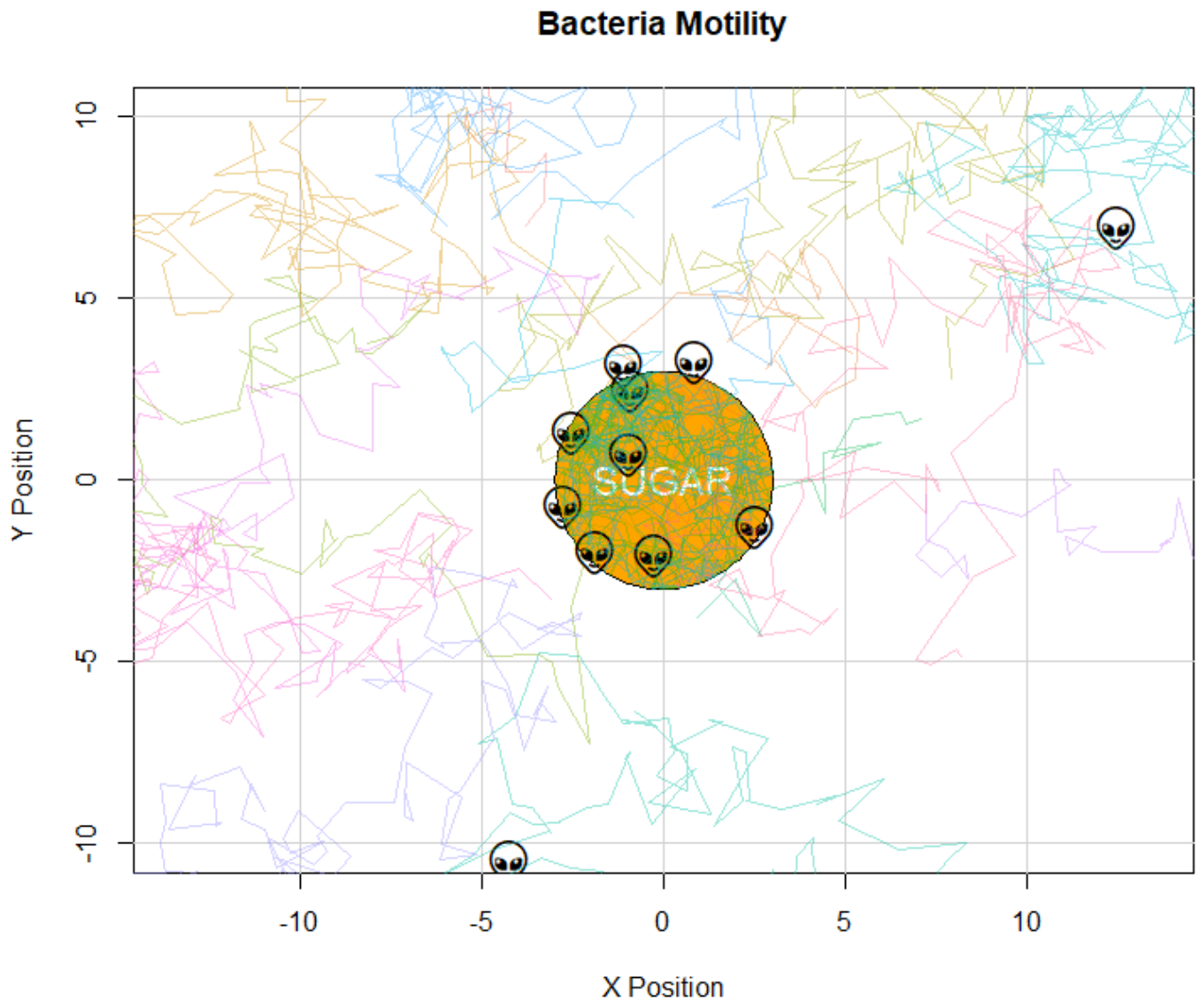
Food for bacteria is usually a simple sugar, such as glucose. Therefore, we will consider food as a circle of sugar of radius 3 centered at $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Furthermore, there exist two options to model the movement of the bacterium once it reaches the circle of sugar. Indeed, you are free to decide that either the bacteria will be staying at the exact position it has reached in the sugar, or it keeps moving randomly, but staying inside this circle.

Bonus points: An extra 10 bonus points will be given if you model the situation where bacteria keep moving inside the circle of sugar after they reach it.

In this problem, you are asked to write a program that allows to visualize the trajectory in a cartesian plane of the 20 bacterium looking for food for the iteration $t = 1, \dots, 150$. The program you will write should produce a similar output to this:



and the final graph should look similar to this:



Hint: the packages `plotrix` and `emojifont` could be helpful for this problem.