Practical 2: A stochastic simulation of passport control queues

This practical is to be completed in groups of 3. What is submitted must be solely the work of the members of the submitting group. Code must not be shared between groups.

This practical is less prescriptive than the previous one. In the previous practical the programming steps were broken down for you. Here you have to do a bit more of that yourself. However the practical should be straightforward if you are up to date with the lecture material, and have put in the effort to understand it properly. The assignment has the 'real world' feature that you have to spend some pencil and paper time translating the problem description into something you can code (albeit the problem description in fact describes what is needed with unusual precision for the real world). Again, coding should use only base R (recommended packages and `ggplot2` are fine, although I don't think any are needed).

Queues are a familiar annoyance, with border queues a more noticeable feature at the UK border since Brexit. When planning port and airport facilities the management of border controls to keep queues acceptable is an important consideration, and for busy and complex ports simulation can provide an invaluable aid to planning. This practical is about writing a function to simulate cars passing through French and then British passport control at a French ferry terminal. For purposes of this project the set up will be a relatively simple one, but still one that is not at all easy to model without simulation. The model is as follows.

1. Interest is in a 2 hour period prior to ferry departure. It is assumed that there are no queues at the start of the period, and that check-in closes 30 minutes before departure so that no new cars arrive in the final half hour.

2. The simulation will be run with a time step of one second: i.e. each second of the 2 hour period the model is updated.

3. There are m_f french passport control stations followed by m_b british ones. Each station has its own queue (initially empty), but the british station queues can only get to 20 cars long before space runs out. When this happens no car can move on from a french station until a space again becomes available on at least one british queue.

4. Cars arrive at the french stations randomly at an average rate of one every 10 seconds, so that there is a probability of 0.1 of an arrival each second. You can neglect the probability of 2 or more cars arriving in the same second.

5. On arrival a car always joins the shortest available french queue.

6. A french station starts processing the next car in its queue, if there is one, as soon as the previous car has left. This processing then takes a random amount of time, modelled as uniform between `tmf` and `tmf+trf` (`tmf` and `trf` are model constants – see below).

7. As soon as a car completes processing at a french station, it can move on to a british queue, provided there is one available that is not full (i.e. that has less than 20 cars). Otherwise it has to wait at the french station until it can move on.

8. A car moving on to a British queue always selects the shortest one available.

9. As soon as the car moves on from a french station that station can start processing the next car in its queue, if there is one.

10. British stations process cars in the same way as french stations, but processing takes a random time that is uniform between `tmb` and `tmb+trb`. Cars can leave british stations as soon as they are processed (i.e. there is no possibility of being blocked by a queue beyond that point).

Write a function to simulate from this model:
`qsim(mf=5,mb=5,a.rate=.1,trb=40,trf=40,tmb=30,tmf=30,maxb=20)`
where the argumets are as follows:

`mf` is m_f.

`mb` is m_b.

`a.rate` is the probability of a car arriving each second.

`trb`, `trf`, `tmb` and `tmf` are constants as given above.

`maxb` is the maximum british queue length (per station).

The function should return a list with 3 named elements:

`nf` a vector giving the average length of the french queues, for each simulation second.

`nb` a vector giving the average length of the british queues, for each simulation second.

`eq` a vector giving the average expected waiting time for a car at the start of the french queue for each second of the simulation (this can be worked out from the queue lengths and average processing times per station).

You must stick exactly to the specification of function inputs and outputs, including using the given names, in order that the work can be marked efficiently.

The default simulation parameters are set up so that the average rate of processing passports by both the French and the British exactly matches the arrival rate. Suppose that interest is in examining what happens if the British handling rate falls behind this rate, by a few seconds per car.

You should write code to use your `qsim` function to produce a 4 panel 2 row, 2 column plot in which the top row shows a plot of how the French and British queue lengths are changing over time and a second plot of the how the expected queuing time changes over time, for a simulation with the default parameters. The second row should show equivalent plots when `mb` (the minimum British handling time) is set to 40 seconds.

Finally write code to estimate the probability of at least one car missing the ferry departure (i.e. still being in the queue at the end of the simulation). Use 100 runs of you `qsim` function to so this.

One piece of work - the text file containing your commented R code - is to be submitted for each group of 3 on Learn by 12:00 20th October 2023. If you are on statistical programming, please name the file `SGxx.r` where `xx` is your group number. If you are on extended statistical programming please name the file `EGxx.r` where `xx` is your group number.

You may be asked to supply an invitation to your github repo, so ensure this is in good order. No extensions are available on this course, because of the frequency with which work has to be submitted. So late work will automatically attract a penalty (of 100% after work has been marked and returned). Technology failures will not be accepted as a reason for lateness (unless it is provably the case that Learn was unavailable for an extended period), so aim to submit ahead of time.

Marking Scheme: Full marks will be obtained for code that:

1. does what it is supposed to do, and has been coded in R approximately as indicated (that is marks will be lost for simply finding a package or online code that simplifies the task for you).
2. is carefully commented, so that someone reviewing the code can easily tell exactly what it is for, what it is doing and how it is doing it without having read this sheet, or knowing anything else about the code in advance. Note that *easily tell* implies that the comments must also be as clear and *concise* as possible. You should assume that the reader knows basic R, but not that they know exactly what every function in R does.
3. is well structured and laid out, so that the code itself, and its underlying logic, are easy to follow.
4. is reasonably efficient. As a rough guide one simulation run should take less than a second to run - longer than that and something is probably wrong.
5. includes a comment briefly discussing the implications of small extra delays in British checking.
6. was prepared collaboratively using git and github in a group of 3.
7. has been tidied up so that it is a 'finished product' - no commented out experimental code, code purely for your own checking purposes, or printing out of large objects. Put functions first, and code using the functions after that.