Practical 2: The prisoner problem simulation

This practical uses stochastic simulation to investigate a somewhat surprising probability puzzle. Along the way it makes an important statistical point about how badly wrong it is possible to go by assuming events, or random variables, are independent when they are not.

The puzzle set up is as follows ('number' means 'natural number' here):

- $2n$ prisoners each have a unique prisoner number from 1 to $2n$.
- The prison contains a room in which there are $2n$ boxes, each with a unique number from 1 to $2n$ painted on its lid.
- $2n$ cards, each printed with a unique number from 1 to $2n$, are randomly placed one in each box.
- The prisoners have the task of finding the card with their number on it by opening a maximum on n boxes.
- After each prisoner's go, the room is returned exactly to its original state and the prisoner is not allowed to communicate with prisoners yet to have their go.
- If all prisoners succeed in finding their number, then they all go free.

One of the following strategies has a surprisingly high probability of allowing all prisoners to go free:

1. The prisoner starts at the box with their number on it, opens it and reads the number on the card: k, say. If k is not their prisoner number, they go to box number k, open it and repeat the process until they have either found the card with their number on it, or opened n boxes without finding it.
2. As strategy 1, but starting from a randomly selected box.
3. They open n boxes at random, checking each card for their number.

Your task is to write well structured, well commented, code to estimate the probabilities of success for a single prisoner under each strategy (that is, what is the probability that one prisoner finds their number), and then for all prisoners to succeed so that they are freed. You will then write code to estimate a probability distribution that is key to understanding the surprising result.

1. Write a function `Pone` to estimate the probability of a single prisoner succeeding in finding their number. The function takes arguments `n`, `k` (the prisoner's number), `strategy` (1, 2 or 3) and `nreps`. `nreps` is the number of replicate simulations to run in order to estimate the probability (10000 is a reasonable default). The function should return the probability estimate.
2. Write a function `Pall` to estimate the probability of all prisoners finding their number, so that all are released. This has the same arguments as `Pone`, except for `k`, of course. Make sure that your code is structured so that you do not un-necessarily repeat code between the two functions, and it is clear to read.
3. Provide example code using your functions to estimate the individual and joint success probabilities under each strategy for $n = 5$ and for $n = 50$.
4. Include in your comments brief remarks on what is surprising about the results.
5. The surprisingly good strategy works because of the presence of loops in the sequence of cards in boxes. A loop occurs when some card in the sequence of opened boxes (under strategy 1 or 2) has the number of the first box opened. Suppose that u is a $2n$-vector, such that $u[i]$ gives the number of the card in box number i. Formally a loop occurs when $u[u[\cdots u[u[k]]\cdots]] = k$, where the length of loop is given by the depth of nesting. For example $u[k] = k$ is a loop of length 1, $u[u[k]] = k$ is a loop of length 2, $u[u[u[k]]] = k$ is a loop of length 3, and so on.

   Write a function `dloop` to estimate, by simulation, the probability of each loop length from 1 to $2n$ occurring at least once in a random shuffling of cards to boxes. The functions should take `n` and `nreps` as arguments, and return a $2n$-vector of probabilities.

6. Provide example code using `dloop` to estimate the probabilities for $n = 50$, assessing the probability that there is no loop longer than 50 in a random reshuffling of cards to boxes, and also visualising the probabilities sensibly.

As a group of 3 you should aim to produce well commented[1], clear and efficient code for the task. The code should be written in a plain text file called `proj2.r` and is what you will submit. Your solutions should use only the functions available in base R (or packages supplied with base R - no other packages). The work must be completed in your work group of 3, which you must have arranged and registered on Learn, and collaborative code development must be done using a github repo set up for this purpose.

The first comment in your code should list the names and university user names of each member of the group. The second comment should give the address of your github repo, which should be made public *after* the hand in deadline. The third comment **must** give a brief description of what each team member contributed to the project, and roughly what proportion of the work was undertaken by each team member. Contributions never end up completely equal, but you should aim for rough equality, with team members each making sure to 'pull their weight', as well as not unfairly dominating[2]. Any team member that did not take an active part in actually writing code for project 1, should ensure they do so this time.

Your code file should not refer to this sheet, and should be sufficiently well commented that someone reading it can tell what it is about and the strategies you are using without reference to this sheet. Makes sure that your code takes no more than abut one minute to run (at most two).

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 Friday 21st October 2022. Format the code and comments tidily, so that they display nicely on an 80 character width display, without line wraps (or only occasional ones). 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 hefty 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. 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 the whole code should take less than a minute to run - much longer than that and something is probably wrong.
5. was prepared collaboratively using git and github in a group of 3.
6. contains no evidence of having been copied, in whole or in part, from other students on this course, students at other universities (there are now tools to help detect this, including internationally), online sources etc.
7. includes the comment stating team member contributions.

Individual marks may be adjusted within groups if contributions are widely different.

[1] Good comments give and overview of what the code does, as well as line-by-line information to help the reader understand the code. Generally the code file should start with an overview of what the code in that file is about, and a high level outline of what it is doing. Similarly each function should start with a description of its inputs outputs and purpose plus a brief outline of how it works. Line-by-line comments aim to make the code easier to understand in detail.

[2] all team members must have git installed and use it - not doing so will count against you if there are problems of seriously unequal contributions