Project 2

Will Graham; Richelle Lee; Robin Lin

2022-10-21

### Group Contributions:  
# Instead of splitting tasks between our group, we worked collaboratively on each step of the process.  
# Github Repository Address:  
# https://github.com/willlewisgraham/Statistical-Programming-2022.git  
# Contributions were 1/3 per team member.

Pone <- function(n, k, strategy, nreps = 10000){  
 # The ‘Pone’ function aims at estimating the probability of an individual   
 # prisoner finding their number based on the number of  
 # prisoners and the types of strategies they adopt.   
 #  
 # ‘n’ represents the maximum number of boxes the prisoner could open.   
 #  
 # ‘k’ represents the number of a particular prisoner. It is worth noting that   
 # the probability of the k-th prisoner finding the card number matching their  
 # number will not be affected by the selection of ‘k’.   
 #  
 # ‘strategy’ could take on 3 values: 1, 2, or 3, representing different kinds   
 # of strategies.   
 #  
 # In strategy 1, the prisoner selects a box with the same number as their   
 # own, and reads the number on the card in the box. If   
 # the number is not   
 # theirs, they will go to the box with the number on the card. This process   
 # repeats until they find the card   
 # with their number on it, or they open   
 # n boxes.  
 #  
 # In strategy 2, the prisoner selects a random box, and then selects   
 # subsequent boxes with the same method as strategy 1.  
 #  
 # In strategy 3, the prisoner opens n boxes randomly, checking the cards for   
 # their number.  
 #  
 # ‘nreps’ represents the number of simulations being conducted. In our   
 # experiment, the process is being simulated for 10000 times.  
 #  
 # The return of the function should be the rate of individual successes in   
 # the 10000 simulations.  
   
   
 successes <- 0  
 static\_choices <- sample(1 : (2 \* n), n, replace = FALSE) # For strategy 3, generate all the random choices that a prisoner will make when selecting their next cards  
   
 for (j in 1 : nreps){  
 i <- 1  
 cards <- sample(1 : (2 \* n), 2 \* n, replace = FALSE)  
 # Determine next box based on the input strategy  
 next\_box <- switch(strategy, k, sample(1 : (2 \* n), 1), static\_choices[i])   
 while (i <= n){  
 card <- cards[next\_box]  
 if (card != k){  
 i <- i + 1  
 next\_box <- switch(strategy, card, card, static\_choices[i])  
 } else {  
 successes <- successes + 1 # If the prisoner finds the card with their number on it, record a success  
 i <- n + 1  
 }  
 }  
 }  
 return (successes / nreps)   
}

Pall <- function(n, strategy, nreps = 10000){  
 # Simulates nreps number of trials to estimate the probability that 2n   
 # prisoners successfully find cards with their numbers  
 # within a maximum of n box selections using an input strategy (1, 2, or 3).   
 # The distribution of cards among boxes is assumed to be identical for all   
 # prisoners part of the same trial.  
   
 trial\_successes = 0  
 for (i in 1 : nreps){  
 static\_choices <- matrix(0, 2 \* n, n)  
 # For strategy 3, generate all the random choices that prisoners will make   
 # when selecting their next cards:  
 for (k in 1 : (2 \* n)){   
 static\_choices[k,] <- sample(1 : (2 \* n), n, replace = FALSE)  
 }  
 prisoner\_successes <- matrix(FALSE, n, 2\* n)  
 cards <- sample(1 : (2 \* n), 2 \* n, replace = FALSE)  
   
 prisoner\_current\_card <- switch(strategy, cards, sample(1 : (2 \* n), 2 \* n, replace = FALSE), static\_choices[,1])   
 # Determine the prisoners' next cards based on the input strategy  
 prisoner\_successes[1,] <- prisoner\_current\_card == 1 : (2 \* n)   
 # Record the results from the prisoners’ initial positions  
 for (j in 2 : n){  
 prisoner\_current\_card <- switch(strategy, cards[prisoner\_current\_card], cards[prisoner\_current\_card], static\_choices[,j])  
 prisoner\_successes[j,] <- prisoner\_current\_card == 1 : (2 \* n)   
 # Record the results after the prisoners make j box choices  
 }  
 prisoner\_results <- apply(prisoner\_successes, 2, sum)  
 if (!(0 %in% prisoner\_results)){ # If no prisoner failed on every turn, the trial was a success  
 trial\_successes <- trial\_successes + 1  
 }   
 }  
 return(trial\_successes / nreps)  
}

# The following results are counterintuitive. Even though strategies 1 and 3   
# appear to have the same success rate at an individual level, strategy 1   
# massively overperforms the other 2 strategies at a joint level. This   
# indicates that the success rates of individual prisoners are not independent   
# from each other in the case of strategy 1; i.e., given that 1 prisoner   
# succeeds with strategy 1, the probability of other prisoners also succeeding   
# is increased.   
   
for (i in c(5, 50)){  
 for (j in 1 : 3){  
 cat("When n = ", i, " and strategy = ", j, ", the estimated probability of an individual prisoner finding their number is ", round(Pone(n = i, k = 1, strategy = j), 2),"\n", sep = "")  
 }  
}

## When n = 5 and strategy = 1, the estimated probability of an individual prisoner finding their number is 0.5  
## When n = 5 and strategy = 2, the estimated probability of an individual prisoner finding their number is 0.41  
## When n = 5 and strategy = 3, the estimated probability of an individual prisoner finding their number is 0.5  
## When n = 50 and strategy = 1, the estimated probability of an individual prisoner finding their number is 0.51  
## When n = 50 and strategy = 2, the estimated probability of an individual prisoner finding their number is 0.37  
## When n = 50 and strategy = 3, the estimated probability of an individual prisoner finding their number is 0.51

for (i in c(5, 50)){  
 for (j in 1 : 3){  
 cat("When n = ", i, " and strategy = ", j, ", the estimated probability of all prisoners finding their numbers is ", round(Pall(n = i, strategy = j), 4),"\n", sep = "")  
 }  
}

## When n = 5 and strategy = 1, the estimated probability of all prisoners finding their numbers is 0.3569  
## When n = 5 and strategy = 2, the estimated probability of all prisoners finding their numbers is 5e-04  
## When n = 5 and strategy = 3, the estimated probability of all prisoners finding their numbers is 8e-04  
## When n = 50 and strategy = 1, the estimated probability of all prisoners finding their numbers is 0.3088  
## When n = 50 and strategy = 2, the estimated probability of all prisoners finding their numbers is 0  
## When n = 50 and strategy = 3, the estimated probability of all prisoners finding their numbers is 0

dloop <- function(n, nreps = 10000){  
 # The dloop function computes the probabilities of each loop length (1 to 2n)  
 # occurring at least once in a random shuffle of 2n (where n is the max # of   
 # boxes to be opened) cards to 2n boxes.  
   
   
 freq <- matrix(0, nreps, 2 \* n)  
 for (i in 1 : nreps){  
 u <- sample(1 : (2 \* n), 2 \* n, replace = FALSE)  
 cards <- u  
 u <- u[u] # Simulate the next card in a sequence  
 repeated\_numbers <- rep(0, (2 \* n))  
 for (len\_loop in 1 : (2 \* n)){  
 if( sum(u == cards & !( u %in% repeated\_numbers)) > 0) { # Checks the occurrence of a loop length  
 freq[i, len\_loop] <- 1   
 }  
 # Current batch of card numbers that were observed in the preceding sequence  
 current\_repeated\_numbers <- u[u == cards]  
 repeated\_numbers[current\_repeated\_numbers] <- current\_repeated\_numbers  
 u <- cards[u]  
 }  
 }  
 freq\_new <- apply(freq, 2, sum)  
 prob <- freq\_new / nreps  
 return(prob)  
}

loop\_len\_prob <- dloop(50)  
cat("For n = 50, the probability that there is no loop longer than 50 in a random reshuffling of cards to boxes is", (1 - round(sum(loop\_len\_prob[51 : 100]), 2)))

## For n = 50, the probability that there is no loop longer than 50 in a random reshuffling of cards to boxes is 0.32

barplot(loop\_len\_prob, xlab = "Loop Length", ylab = "Probability of at least one occurrence", main = "Loop Length Probability Distribution", names.arg = c(1 : 100), col = c(rep("grey", 50), rep("red", 50)), sub = "Red bars represent the probability of observing at least one loop of length 50 or greater")

