Parallel Programming

Intro to R (Vectors)

Program #2 – The proof that Marilyn Vos Savant was right!

Per our discussion on probability we are going to use some R code (I found someone has solved the Monty Hall Problem in R!), and prove using our data analysis tools that she was right and at what game interation does her solution become exactly correct. Switching = 2/ 3 or 67% and Non-Switching= 1/3 or 33%.

So here’s the code and I added comments to explain it:

#Monty Hall Problem in R Code

#First a vector for our doors

doors <- c("A", "B", "C")

#Then an object to store all the info generated by the loop

xdata = c( )

#The loop - simulate a game at a time

for (i in 1:10000)

{

#place the prize behind a door sample ( ) function index

#this randomly assorted vector with [1]

prize <- sample(doors)[1]

#now simualate a pic if prize and pick are the same gamer got it right first

pick <- sample(doors)[1]

#Game master picks a door other than pick and one that doesn't have the prize

open <- sample(doors[which (doors != pick & doors != prize)])[1]

#The which command returns a number (in above) it is the position of the

#vector door that is not equal to pick & not equal to prize

#which command will give you the number to index the doors vector

#It's possible that prize and pick are equal so use [1] to give us just one value

#Sample() is used to randomize this value

#We have the door picked under no-switching now simulate another participant that switched doors

switchyes <- doors[which(doors != pick & doors != open)]

#Basically - simulate switching by taking one element of the vector doors which was not the original

#pick and not the door that was opened

#Using two if statements find out to see who wins

#Switching or non switching

if(pick == prize)

{

xdata = c(xdata, "noswitchwin")

}

if(switchyes == prize)

{

xdata = c(xdata, "switchwin")

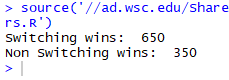
}

}

cat("Switching wins: ", length(which(xdata == "switchwin")),"\n")

cat("Non Switching wins: ", length(which(xdata == "noswitchwin")))

Currently you should get the output similar to this:



This is running the loop 1000 times. As we change the loop value there is a point when we are closes to the 2/3 and 1/3 probability calculations.

Your job is to find the # of loops that approximates the closest 2/3 and 1/3 values. You will make multiple attempts going above and beyond the final values. Store all your data into a vectors.

Your loop vector is best if it looks like this: c(10,20,30,40,50,60,70,80,90,100,110)

I used swClose = .6666 and nonswClose = .3333

Provide a similar output as mine:

