Chem/Stat3240: Homework 6
Matlab

October 15, 2015

The Monty Hall Problem is described at the following link:
http://en.wikipedia.org/wiki/Monty_Hall_problem

1. Write a function
   The input

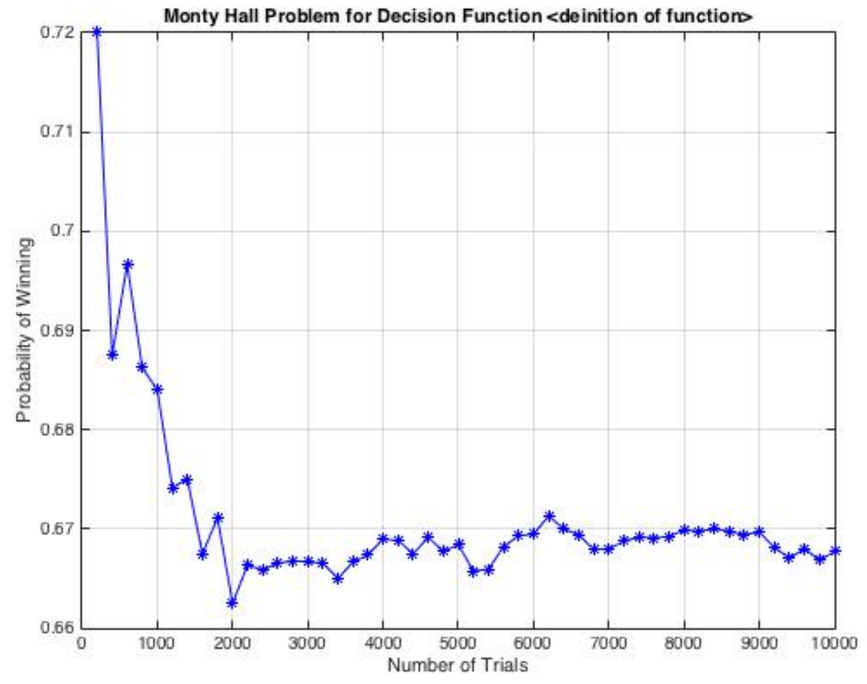
   `[ntrialWins, ntrialVec]=montyHall(ntrials,decideSwitch,plotsOn)`

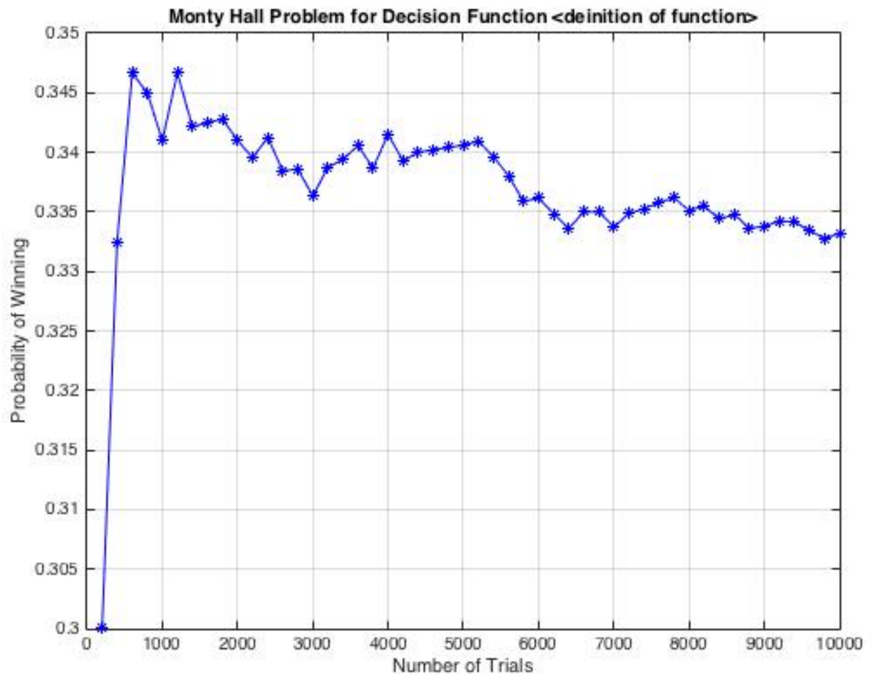
   to simulate the Monty Hall problem using 4 different ways to model the decision to switch doors. The input `ntrials` is the total number of games to run, `decideSwitch` is a function used to decide whether to switch doors after the initial door pick, and `plotsOn` is a character string to determine whether to produce and save a plot of the empirically computed probabilities of winning the prize as a function of the number of trials, as shown in the plots below. The outputs `ntrialWins` and `ntrialVec` are vectors of the computed probabilities of winning, and the associated trial numbers at which those probabilities are computed, the same as used to create the plots below.

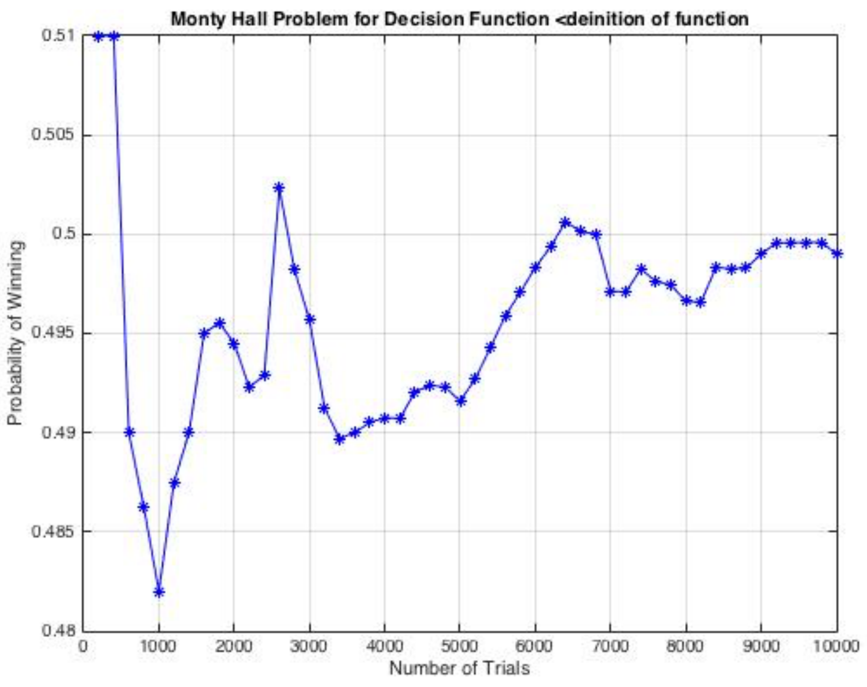
   The `montyHall` function will simulate `ntrials` games of a contestant picking one of three doors, with each door being equally likely. With the door picked, the function will simulate the decision to switch doors by calling the input function `decisionSwitch` and comparing its output to the threshold 0.5 in order to determine whether to switch doors. You will write four anonymous functions to model how this decision might be made, with names '`always`', '`never`', '`flip`', and '`whyNot`'. Respectively, these functions will model always switching doors, never switching doors, flipping a fair coin to decide, and the fourth function will model risking-taking as a normal distribution with positive mean value. The tests for the montyHall function on Cody already defines these functions, which is why those test are not visible.

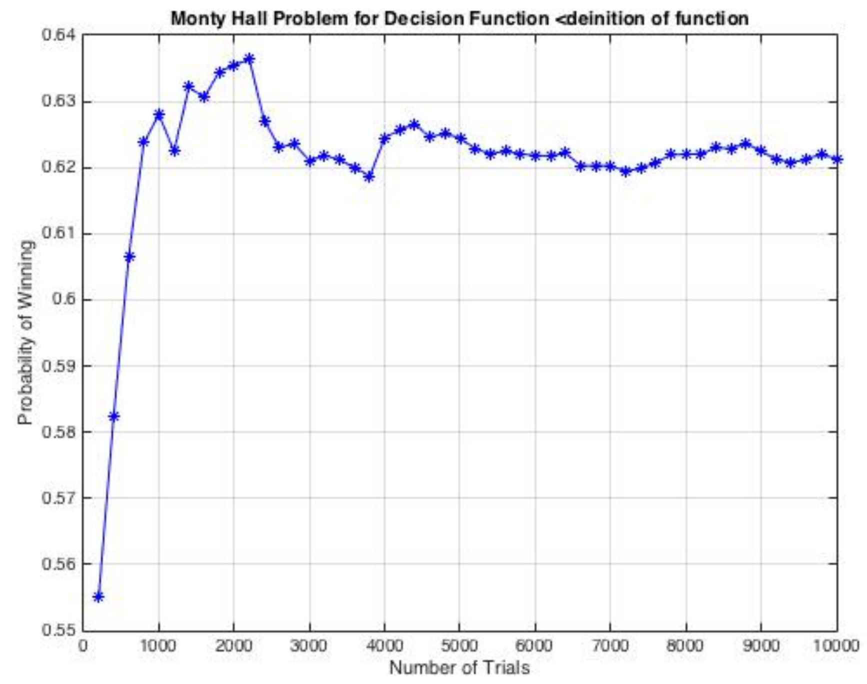
For a given game, once the door is picked and the decision whether to switch doors is made, these are passed as inputs to a function
`[win] = chooseDoor(door, switch_door)`
to determine if the contestant won the prize. The `door` input is an integer from 1 to 3 and the `switch_door` input is either 0 (no switch) or 1 (switch). The function will generate a random integer to simulate which of three doors the prize is behind for that trial. Then it will use a series of compound conditionals based in the two inputs to determine whether the prize is won for that trial and assign the output win to be either 1 or 0.

For a given number of games `ntrials`, the function will calculate the empirical probability of winning the prize for a given decision function when the number of trials is a multiple of `nSample`. To calculate what `nSample` should be, you need to determine how large and integer `n` has to be such that `ntrials/n` < 50. That sampling interval will insure you compute and plot approximately 50 emprical probabilities, no mater how large `ntrials` might be. Example plots are shown below.


Monty Hall Problem for Decision Function <deinition of function>
Probability of Winning
Number of Trials
0.72
0.71
0.7
0.69
0.68
0.67
0.66
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


Monty Hall Problem for Decision Function <deinition of function>
Probability of Winning
Number of Trials
0.35
0.345
0.34
0.335
0.33
0.325
0.32
0.315
0.31
0.305
0.3
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000





Use the `func2tr` command to indicate in the plot title which decisionSwitch function was used.

The function montyHall should verify that the user will win approximately 2/3 of the trials if they always switch from their initial choice of the door.

Once your `montyHall` function has passed the Cody tests, you can use it with the myMonteHallTest.m and write in the definitions for the functions `always`, `never`, `flip`, and `whyNot` to verify your function definitions work.

Submit your function files `montyHall.m` with `chooseDoor` included as a subfunction to the Cody site. Submit those along with `myMonteHallTest`, and the pdfs of the four plots shown above to Collab site as well.

Addendum:

1. Functions should always be called with an argument list, even if its empty. So when you invoke the `decisionSwitch` function in `montyHall`, you should use `decisionSwitch()` (Matlab) or `decisionSwitch[]` (Mathematica). The functions `always`, `never, flip`, and `whyNot` have no input arguments, so you should define them with an empty argument list. They return just one numeric value.
2. For testing purposes, its probably easier to put all your decision functions in a script file `decisionSwitch.m`, that you can run the script to set the various definitions for input to `montyHall`.