**Problem Description :** Implement Monty Hall Paradox using large scale Random Variate generation.

**Learning Objectives**

Know whether it is better to switch or stay

Explain why the probabilities are not equal.

**Theory:**

The Monty Hall problem is a well-known puzzle in probability derived from an American game show, Let’s Make a Deal. (The original 1960s-era show was hosted by Monty Hall, giving this puzzle its name.) Intuition leads many people to get the puzzle wrong, and when the Monty Hall problem is presented in a newspaper or discussion list, it often leads to a lengthy argument in letters-to-the-editor and on message boards.

The game is played like this:

1. The game show set has three doors. A prize such as a car or vacation is behind one door, and the other two doors hide a valueless prize called a Zonk; in most discussions of the problem, the Zonk is a goat.
2. The contestant chooses one door. We’ll assume the contestant has no inside knowledge of which door holds the prize, so the contestant will just make a random choice.
3. The smiling host Monty Hall opens one of the other doors, always choosing one that shows a goat, and always offers the contestant a chance to switch their choice to the remaining unopened door.
4. The contestant either chooses to switch doors, or opts to stick with the first choice.
5. Monty calls for the remaining two doors to open, and the contestant wins whatever is behind their chosen door.

Let’s say a hypothetical contestant chooses door #2. Monty might then open door #1 and offer the chance to switch to door #3. The contestant switches to door #3, and then we see if the prize is behind #3.

The puzzle is: what is the best strategy for the contestant? Does switching increase the chance of winning the car, decrease it, or make no difference?

The best strategy is to make the switch. It’s possible to analyze the situation and figure this out, but instead we’ll tackle it by simulating thousands of games and measuring how often each strategy ends up winning.

**Approach**

Simulating one run of the game is straightforward. A simulate() function that uses Python’s random module to pick which door hides the prize, the contestant’s initial choice, and which doors Monty chooses to open has to be written. An input parameter controls whether the contestant chooses to switch, and simulate() will then return a Boolean telling whether the contestant’s final choice was the winning door.

Part of the reason the problem fools so many people is that in the three-door case the probabilities involved are 1/3 and 1/2, and it’s easy to get confused about which probability is relevant. Considering the same game with many more doors makes reasoning about the problem much clearer, so the number of doors is made as a configurable parameter of the simulation script.

**CODE**

#!/usr/bin/env python3

"""Simulate the Monty Hall problem.

"""

import argparse, random

def simulate(num\_doors, switch, steps):

"""(int, bool): bool

Carry out the game for one contestant. If 'switch' is True,

the contestant will switch their chosen door when offered the chance.

Returns a Boolean value telling whether the simulated contestant won.

"""

# Doors are numbered from 0 up to num\_doors-1 (inclusive).

# Randomly choose the door hiding the prize.

winning\_door = random.randint(0, num\_doors-1)

if steps:

print('Prize is behind door {}'.format(winning\_door+1))

# The contestant picks a random door, too.

choice = random.randint(0, num\_doors-1)

if steps:

print('Contestant chooses door {}'.format(choice+1))

# The host opens all but two doors.

closed\_doors = list(range(num\_doors))

while len(closed\_doors) > 2:

# Randomly choose a door to open.

door\_to\_remove = random.choice(closed\_doors)

# The host will never open the winning door, or the door

# chosen by the contestant.

if door\_to\_remove == winning\_door or door\_to\_remove == choice:

continue

# Remove the door from the list of closed doors.

closed\_doors.remove(door\_to\_remove)

if steps:

print('Host opens door {}'.format(door\_to\_remove+1))

# There are always two doors remaining.

assert len(closed\_doors) == 2

# Does the contestant want to switch their choice?

if switch:

if steps:

print('Contestant switches from door {} '.format(choice+1), end='')

# There are two closed doors left. The contestant will never

# choose the same door, so we'll remove that door as a choice.

available\_doors = list(closed\_doors) # Make a copy of the list.

available\_doors.remove(choice)

# Change choice to the only door available.

choice = available\_doors.pop()

if steps:

print('to {}'.format(choice+1))

# Did the contestant win?

won = (choice == winning\_door)

if steps:

if won:

print('Contestant WON', end='\n\n')

else:

print('Contestant LOST', end='\n\n')

return won

def main():

# Get command-line arguments

parser = argparse.ArgumentParser(

description='simulate the Monty Hall problem')

parser.add\_argument('--doors', default=3, type=int, metavar='int',

help='number of doors offered to the contestant')

parser.add\_argument('--trials', default=10000, type=int, metavar='int',

help='number of trials to perform')

parser.add\_argument('--steps', default=False, action='store\_true',

help='display the results of each trial')

args = parser.parse\_args()

print('Simulating {} trials...'.format(args.trials))

# Carry out the trials

winning\_non\_switchers = 0

winning\_switchers = 0

for i in range(args.trials):

# First, do a trial where the contestant never switches.

won = simulate(args.doors, switch=False, steps=args.steps)

if won:

winning\_non\_switchers += 1

# Next, try one where the contestant switches.

won = simulate(args.doors, switch=True, steps=args.steps)

if won:

winning\_switchers += 1

print(' Switching won {0:5} times out of {1} ({2}% of the time)'.format(

winning\_switchers, args.trials,

(winning\_switchers / args.trials \* 100 ) ))

print('Not switching won {0:5} times out of {1} ({2}% of the time)'.format(

winning\_non\_switchers, args.trials,

(winning\_non\_switchers / args.trials \* 100 ) ))

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Code Discussion**

The command-line arguments are parsed using the argparse module, and the resulting values are passed into the simulate() function.

When there are num\_doors doors, simulate() numbers the doors from 0 up to num\_doors-1. random.randint(a, b)() picks a random integer from the range a to b, possibly choosing one of the endpoints, so here we use random.randint(0, num\_doors-1)().

To figure out which doors the host will open, the code makes a list of the currently closed doors, initially containing all the integers from 0 to num\_doors-1. Then the code loops, picking a random door from the list to open. By our description of the problem, Monty will never open the contestant’s door or the one hiding the prize, so the loop excludes those two doors and picks a different door. The loop continues until only two doors remain, so Monty will always open num\_doors-2 doors.

To implement the contestant’s switching strategy, we take the list of closed doors, which is now 2 elements long, and remove the contestant’s current choice. The remaining element is therefore the door they’re switching to.

**Solution/OUTPUT**

The simulation script is executed from the command line.

**Default case: no arguments**

**the script will use three doors and run 10,000 trials of both the switching and not-switching strategies**

mhr@mhr-desktop:~/Desktop$ python3 montyhall.py

Simulating 10000 trials...

Switching won 6770 times out of 10000 (67.7% of the time)

Not switching won 3333 times out of 10000 (33.33% of the time)

**Case 1:You can supply --doors=100 to use 100 doors**

mhr@mhr-desktop:~/Desktop$ python3 montyhall.py --doors=100

Simulating 10000 trials...

Switching won 9888 times out of 10000 (98.88% of the time)

Not switching won 102 times out of 10000 (1.02% of the time)

**Case 2:You can supply --doors=100 to use 100 doors and --trials=1000 to run a smaller number of trials.**

mhr@mhr-desktop:~/Desktop$ python3 montyhall.py --doors=100 --trials=1000

Simulating 1000 trials...

Switching won 990 times out of 1000 (99.0% of the time)

Not switching won 14 times out of 1000 (1.4000000000000001% of the time)

**Case 3:You can supply --doors=3 to use 3 doors and --trials=4 to run a smaller number of trials and --steps switch, the simulator will print out each step of the game**

mhr@mhr-desktop:~/Desktop$ python3 montyhall.py --steps --doors=3 --trials=4

Simulating 4 trials...

Prize is behind door 2

Contestant chooses door 2

Host opens door 3

Contestant WON

Prize is behind door 3

Contestant chooses door 1

Host opens door 2

Contestant switches from door 1 to 3

Contestant WON

Prize is behind door 2

Contestant chooses door 1

Host opens door 3

Contestant LOST

Prize is behind door 1

Contestant chooses door 2

Host opens door 3

Contestant switches from door 2 to 1

Contestant WON

Prize is behind door 1

Contestant chooses door 2

Host opens door 3

Contestant LOST

Prize is behind door 2

Contestant chooses door 1

Host opens door 3

Contestant switches from door 1 to 2

Contestant WON

Prize is behind door 3

Contestant chooses door 2

Host opens door 1

Contestant LOST

Prize is behind door 3

Contestant chooses door 1

Host opens door 2

Contestant switches from door 1 to 3

Contestant WON

Switching won 4 times out of 4 (100.0% of the time)

Not switching won 1 times out of 4 (25.0% of the time)

**Other cases students can try:**

mhr@mhr-desktop:~/Desktop$ python3 montyhall.py --steps --doors=2 --trials=4

Simulating 4 trials...

Prize is behind door 2

Contestant chooses door 2

Contestant WON

Prize is behind door 1

Contestant chooses door 1

Contestant switches from door 1 to 2

Contestant LOST

Prize is behind door 2

Contestant chooses door 2

Contestant WON

Prize is behind door 1

Contestant chooses door 2

Contestant switches from door 2 to 1

Contestant WON

Prize is behind door 1

Contestant chooses door 2

Contestant LOST

Prize is behind door 1

Contestant chooses door 1

Contestant switches from door 1 to 2

Contestant LOST

Prize is behind door 1

Contestant chooses door 2

Contestant LOST

Prize is behind door 2

Contestant chooses door 2

Contestant switches from door 2 to 1

Contestant LOST

Switching won 1 times out of 4 (25.0% of the time)

Not switching won 2 times out of 4 (50.0% of the time)

**Other cases students can try:**

mhr@mhr-desktop:~/Desktop$ python3 montyhall.py --steps --doors=1 --trials=4

Simulating 4 trials...

Prize is behind door 1

Contestant chooses door 1

Traceback (most recent call last):

File "montyhall.py", line 110, in <module>

main()

File "montyhall.py", line 92, in main

won = simulate(args.doors, switch=False, steps=args.steps)

File "montyhall.py", line 46, in simulate

assert len(closed\_doors) == 2

AssertionError

**To Do:**

Remove the lines in the code that are marked in red(assertion) and ask the students to check for input with no. of doors <2

Solution:students should write assertion statements for this.

In Similar lines,ask students to write assertion for conditions(when winning door and choice is illegal)

1. winning\_door < num\_doors

2.choice < num\_doors

In the code random.randint() is used to generate possible inputs.

randint() returns random integers from the "discrete uniform" distribution in the "half-open" interval ['low', 'high')

Ask the students to use various other functions which generate specific real-valued distributions like random.gauss(),random.uniform() etc.,

**Learning outcome:**

Random generation of possible inputs

usage of pseudo random number genarators

Monte Carlo simulation

Building probability model

Conditional probability

**References**

1. http://en.wikipedia.org/wiki/Monty\_Hall\_problem
2. <https://en.wikipedia.org/wiki/Random_number_generation>
3. <https://docs.python.org/2/howto/argparse.html>
4. <http://www.youtube.com/watch?v=0W978thuweY> -Monty Hall game
5. <https://www.staff.ncl.ac.uk/mike.cox/PsyFin/ohp11.ppt-> [M onte](https://www.staff.ncl.ac.uk/mike.cox/PsyFin/ohp11.ppt-Monte) Carlo simulation
6. <https://courses.csail.mit.edu/6.042/fall06/lec19.pdf> - probability
7. https://docs.python.org/2/library/random.html