Project 3: Monte Carlo Simulations

Colin Gibbons-Fly

Roulette Wheel

In the game of roulette, a ball rolls around a roulette wheel landing on one of 38 numbers. Eighteen numbers are red, 18 are black, and 2 (0 and 00) are green. A bet of "red" costs \$1 to play and pays out \$5, for a net gain of \$4, if the ball lands on red. Let X be a player's winnings at roulette after one bet of red. What are the expected (average) player's winnings after one bet of red? Is it worth playing this game of chance? Why?

Hint: Use the `np.random.choice()` function with the correct list of probabilities for the different possible outcomes (red, black and green), so that you can simulate the gain from playing one game of roulette wheel. Wrap your code up into a Python function and use a list comprehension to call this function 10,000 times, and generate a sample of the winnings from playing this game once. Then, take the average of this sample.

```python
import numpy as np


def roulette():
  #Set board and odds
  colors = ['red','black','green']
  odds = [18/38, 18/38, 2/38]
  roll = np.random.choice(colors, p=odds)

  #Return 1 for red
  if roll == 'red':
    return 1
  else:
    return 0

#Run simulation
sample = [roulette() for i in range(100000)]
result = np.mean(sample)
```

```
print(f"Odds of hitting red: {result:.2%}")
```

```
Odds of hitting red: 47.37%
```

The Monty Hall Problem

In a popular game show in the 1970s , the host shows you a wall with three closed doors. Behind one of the doors is a fancy car, and behind each of the other two is a goat. The three doors all have equal chances of hiding the car. The host knows exactly what is behind each of the three doors. First, you choose a door without opening it, knowing that after you have done so, the host will open one of the two remaining doors to reveal a goat. When this has been done, you will be given the opportunity to switch doors and win whatever is behind the door you choose.

- What is the probability to win the car if you never switch doors?
- What is the probability to win the car if you always switch doors?

Hint: In the case when you always switch the doors, use the `np.random.choice()` function to simulate your choice of door and also the door that hides the car. Write a function such that after these two simulations the function should return 0 if you don't get the car after the host reveals of the goats, and you switch the door to the last door remaining. The function should return 1 in the case you do get the car after you switch the door to the remaining door when the host opens one of the doors with a goat behind. Then, use a list comprehension to call this function 10,000 times, and compute the probability of winning the car if you always switch doors.

```
import numpy as np

def monty_hall_stay():
    # Setting stage
    doors = np.random.choice(['goat', 'goat', 'car'], size=3, replace=False)

    # First contestant guess
    guess = np.random.choice(3)

    # Host opens door
    host_open = np.random.choice([i for i in range(3) if i != guess and doors[i] != 'car']

    # Returns 1 or 0 depending on guess
    if doors[guess] == 'car':
        return 1
    elif doors[guess] == 'goat':
        return 0

# Run simulation
sample = [monty_hall_stay() for i in range(10000)]
stay_result = np.mean(sample)

print(f"Probability of winning by staying: {stay_result:.2%}")
```

```
Probability of winning by staying: 33.19%
```

```
import numpy as np

def monty_hall_switch():
    # Setting stage
    doors = np.random.choice(['goat', 'goat', 'car'], size=3, replace=False)

    # First contestant guess
    guess = np.random.choice(3)

    # Host opens door
    host_open = np.random.choice([i for i in range(3) if i != guess and doors[i] != 'car']

    # Contestant switches guess
    guess = [i for i in range(3) if i != guess and i != host_open]

    # Returns 1 or 0 depending on guess
    if doors[guess] == 'car':
        return 1
    elif doors[guess] == 'goat':
        return 0

# Run simulation
sample = [monty_hall_switch() for i in range(10000)]
switch_result = np.mean(sample)

print(f"Probability of winning by switching: {switch_result:.2%}")
```

```
Probability of winning by switching: 67.33%
```

The Gambler’s Ruin with a Twist

Player M has \$1 and player N has \$2. Each play gives one of the players \$1 taken from the other. Player M is known to be better than player N, so that M wins $\frac{2}{3}$ of the plays. They play until one is bankrupt. Using Monte Carlo simulation, estimate the probability that player N goes bankrupt.

Hint: Write a variation of the gambler’s ruin simulation from the course notes. The exact answer is $\frac{4}{7} \approx 0.5714$.

```python
import numpy as np

def gamble():
    #Starting amounts
    player_m = 1
    player_n = 2

    #While loop for each player having more than $0
    while 0 < player_m and 0 < player_n:

      #Returns True if Player M wins, False if Player N wins taking into account their odd
        win_m = np.random.choice([True, False], p=[2/3, 1/3])

      #Distributing the player winnings
        if win_m:
            player_m += 1
            player_n -= 1
        else:
            player_m -= 1
            player_n += 1

    #Returning 1 when Player N goes bankrupt
    if player_n == 0:
        return 1
    else:
      return 0

# Run simulation
results = [gamble() for i in range(100000)]
probability_bankrupt = np.mean(results)

print(f"Probability of Player N going bankrupt: {probability_bankrupt:.2%}")
```

```
Probability of Player N going bankrupt: 57.02%
```

Movie Night

You plan a movie night with a friend but you both realize that you will arrive at the movie theater at some random time between 8:00 pm and 8:45 pm. Assume the arrival time, with respect to 8:00 pm, for each of you, is uniformly distributed between 0 and 45 minutes. However, you both agree to wait for 15 minutes only, after you arrive, and then leave the movie theater. Using Monte Carlo simulations, estimate the probability you will see the movie together on that night.

Hint: Write a simulation based on the solution for the "The Hurried Duelers" problem from the course notes.

```python
def movie_night_simulation(N):
    # Independent random arrivals between 0 and 45 minutes
    my_arrival = 45 * np.random.rand(N)
    friend_arrival = 45 * np.random.rand(N)

    # Check if you and your friend will arrive within 15 minutes
    together_count = np.sum(np.abs(my_arrival - friend_arrival) <= 15)

    # Calculate the probability of seeing the movie together
    probability_together = together_count / N

    return probability_together

# Run the simulation
result = movie_night_simulation(100000)

print(f"Estimated probability of seeing the movie together: {result:.2%}")
```

```
Estimated probability of seeing the movie together: 55.61%
```