Python Project 3 - Using Monte Carlo to Simulate Random Processes

This project has two parts. You may choose to do either. If you do both, then extra credit will be awarded. Also you have the option to work with another person with this project. In that case, only one code need be submitted.

The first involves using MC to simulate a game of chance where we "draw" a ball out of a bin of 100 balls with each ball numbered from 1 to 100. One goal is to determine the average amount of money the player has remaining after playing a fixed number of times. In addition, we want to determine the discrete expected value when we simulate playing and show that it is close to the theoretical expected value.

The second involves using MC to estimate the amount of commission that a company expects to have to pay. Here the sales rep's commission is based upon how close their actual sales are to their target sales. We first want to simulate the sales for each rep and compute the total amount of commission. Then we want to do this many times and average the total commissions from all the simulations.

Part I - Using Monte Carlo to simulate a game of chance

Deepnote Practice FIles / Python Project 3 - Monte Carlo - Duplicate Published at Mar 5, 2023 Unlisted

- Player has 49 out of 100 chances of winning so probability is 0.49
- House has 51 out of 100 chances of winning so probability is 0.51

Clearly the house has an edge which we can compute using the expected value. If you bet $100 then the expected value is

> expected value = (100)(0.49) - (100)(0.51) = - 0.02(100) = -2

So with this bet, you should expect to lose $2 per game.

The goal is to approximate your ending funds if you decide you are only going to play a certain number of games. For example, suppose you are only going to play 5 times then we could simulate the game for 5 times. However, this would only give us the remaining funds for one example of playing 5 games. Instead we want to do this many times and average the result.

Assume that the starting funds are 10,000 dollars and we always wager 100 dollars per game.

We solve this problem in the following steps:

- Step 0. Import library and set seed
- Step 1. Define a function which returns True if the player wins and False otherwise.
- Step 2. Define a function which simulates playing the game a given number of times; it uses the function in Step 1 and adjusts the funds available depending if the player wins or loses. Function returns the amount of funds remaining after playing the given number of times and the number of times the player wins/loses.
- Step 3. Write code to call the function in Step 2 for 5 games. Output the funds remaining after 5 games and number of wins/loses.

- Step 4. Modify the code in Step 4 to do this simulation (of playing 5 games) 100 times. Output the average of the funds remaining after 5 games and the probability of winning & losing; output the expected value for this simulation.

The results you get in Step 4 do NOT match our theoretical expected value of losing 2 dolars per game. You are asked to make modification(s) to your code in Step 4 so that your discrete expected value is close to the theoretical value.

```
# import random library and set seed
import random
random.seed(43)
```

```
# Step 1.
# Define a function which randomly draws a ball with an integer between 1 and 100 on it;
# then returns True if Player
# wins and returns False if House wins.
# Add a print statement to function to print random integer and test out function
# Add comment statements to function
#
def ball_draw():
    pick=random.randint(1,100)
    if pick >= 52:
        print(pick)
        return True
    else:
        print(pick)
        return False
```

```
ball_draw()
```

```
5
```

```
False
```

```
# Step 2.
# Create a function which simulates playing the game n_plays times; keep track of number of wins and losses

def ball_draw_n(n):
    fund = 10000
    wins = 0
    losses = 0
    for i in range (n):
        if ball_draw():
            fund += 100
            wins += 1
        else:
            fund -= 100
            losses += 1
    return (fund, wins, losses)
```

Step 3. Now assume that we wager 100 dollars per game, we start with 10,000 dollars. Write code to call the function in Step 2 to simulate playing 5 games. Print out the funds & number of wins and losses remaining after these 5 games.

```
funds, wins, losses = ball_draw_n(5)
print("remaining funds:", funds)
print("wins:", wins)
print("losses:", losses)
```

```
37
90
98
19
60
remaining funds: 10100
wins: 3
losses: 2
```

Step 4. Now simulate playing 5 games 100 times and print out the average amount of funds remaining when you play 5 games. Print out the discrete probability of winning or losing. As before the discrete probability of winning is just the total number of wins over all simulations divided by the total number of games simulated.

```
def simulation(x,y):
    tfunds=0
    twins=0
    tlosses=0
    for i in range(y):
        funds, wins, losses = ball_draw_n(x)
        tfunds += funds
        twins += wins
        tlosses += losses
    avgfunds = tfunds / y
    probwin = twins / (x * y)
    probloss = 1 - probwin
    expected_loss = (10000 - avgfunds) / x / y
    print(f"Average funds remaining after {x} games:" , avgfunds)
    print("Discrete probability of winning:", probwin)
    print("Discrete probability of losing:",probloss)
    print("Expected amount lost per game:", expected_loss)

simulation(5, 100)
```

```
31
41
47
14
51
12
17
2
67
77
10
Average funds remaining after 5 games: 9974.0
Discrete probability of winning: 0.474
Discrete probability of losing: 0.526
Expected amount lost per game: 0.052000000000000005
```

Now this amount of losing over 8 dollars per game doesn't agree with our theoretical expected value of losing about 2 dollars per game. What do we need to do to get a better approximation? Modify the necessary statement(s) in your program and perform another simulation to get around losing 2 dollars per game.

```python
def ball_draw_n(n):
    fund = 10000
    wins = 0
    losses = 0
    for i in range (n):
        if ball_draw():
            fund += 4000
            wins += 1
        else:
            fund -= 4000
            losses += 1
    return (fund, wins, losses)


def simulation(x,y):
    tfunds = 0
    twins = 0
    tlosses = 0
    for i in range(y):
        funds, wins, losses = ball_draw_n(x)
        tfunds += funds
        twins += wins
        tlosses += losses
    avgfunds = tfunds / y
    probwin = twins / (x * y)
    probloss = 1 - probwin
    expected_loss = (10000 - avgfunds) / x / y
    print(f"Average funds remaining after {x} games:" , avgfunds)
    print("Discrete probability of winning:", probwin)
    print("Discrete probability of losing:",probloss)
    print("Expected amount lost per game", expected_loss)



# if we increase the amount of money wagered, from 100 to 4000 per game we get an expected value of close to $

simulation(5,100)
```

```
91
78
54
5
40
58
34
55
16
83
78
14
8
60
54
41
26
26
52
56
16
76
20
96
99
86
Average funds remaining after 5 games: 8880.0
Discrete probability of winning: 0.472
Discrete probability of losing: 0.528
Expected amount lost per game 2.24
```

Part II - Using Monte Carlo for a business application

Suppose that you work at a company which employs sales people who receive a commission on their sales.

The commission is just calculated from the formula

> commission = actual sales * commission rate

The commission rate is based upon how close the actual sales are to the target sales. The rate schedule we use here is:

> - actual sales between 0-90% of target sales ⇒ 2.00% commission rate
> - actual sales between 91-99% of target sales ⇒ 3.00% commission rate
> - actual sales between >=100% of target sales ⇒ 4.00% commission rate

Suppose that your job is to inform the company's Finance Department how much money to budget for commissions. Assume the company has 5 sales people and the information for last year is summarized in the following table. Further **assume that the Target Sales and number of sales people are the same for the coming year**.

Sales person	Target Sales	Actual Sales	Commission Rate	Commission
1	150,000	122,000	2%	2,440
2	200,000	195,000	3%	5,850
3	95,000	82,000	2%	1,640