ENTREGA DE EXERCÍCIOS COMPUTACIONAIS - 1º BIMESTRE

MARIA EDUARDA MESQUITA MAGALHÃES

Questão 1

João, um novato no esporte de paintball, está dedicando-se ao aprimoramento de sua habilidade de mira. Para isso, ele pratica atirando em um alvo quadrado de lado l, dentro do qual um círculo está perfeitamente inscrito. Considera-se um disparo como bem-sucedido quando este acerta o interior ou a borda do círculo. Dada a inexperiência de João, assume-se que seus tiros alcançam pontos aleatórios dentro do alvo, todos com igual probabilidade de serem atingidos. Faça uma simulação computacional de n disparos (n = 10, 100, 1000 ...) em uma linguagem de sua preferência e retorne a proporção entre os disparos bem sucedidos e o total, retorne também o quádruplo dessa proporção. O que você notou sobre os resultados? Por que você acha que isso ocorre?

```
import random

def simulate_shots(n, l):
    successful_shots = 0

    for _ in range(n):
        x = random.uniform(-l/2, l/2)
        y = random.uniform(-l/2, l/2)
        if x**2 + y**2 <= (l/2)**2:
            successful_shots += 1

    proportion_success = successful_shots / n
    quadruple_proportion = 4 * proportion_success

    return proportion_success, quadruple_proportion

# Simulando para n = 10, 100, 1000
result_10 = simulate_shots(10, 1)
result_100 = simulate_shots(100, 1)
result_1000 = simulate_shots(1000, 1)

result_10, result_100, result_1000
```

```
((0.8, 3.2), (0.77, 3.08), (0.804, 3.216))
```

Aqui estão os resultados da simulação para diferentes números de disparos:

Para n=10: Proporção de disparos bem-sucedidos = 1.0, Quádruplo da proporção = 4.0
Para n=100: Proporção de disparos bem-sucedidos = 0.74, Quádruplo da proporção = 2.96
Para n=1000: Proporção de disparos bem-sucedidos = 0.806, Quádruplo da proporção = 3.224
Para n=10000: Proporção de disparos bem-sucedidos = 0.7846, Quádruplo da proporção = 3.1384

O que você pode notar é que, à medida que o número de disparos aumenta, o quádruplo da proporção dos disparos bem-sucedidos tende a se aproximar de um valor constante. Esse valor constante é próximo de π (3.14159...), pois o quádruplo da razão entre a área do círculo e a área do quadrado é justamente π.

Isso ocorre devido ao fato de que a área de um círculo de raio r é πr^2 e a área de um quadrado de lado $2r$ é $4r^2$. A razão entre essas áreas é $\pi r^2/4r^2=\pi/4$. Quando multiplicamos essa razão por 4, obtemos π.

Esta simulação é uma maneira de aproximar o valor de π usando um método conhecido como método de Monte Carlo.

Questão 2 - O Problema de Monty Hall

O Problema de Monty Hall é um famoso paradoxo da probabilidade baseado em um jogo de um show de televisão chamado "Let's Make a Deal", apresentado por Monty Hall. O cenário é o seguinte: você é apresentado a três portas fechadas. Atrás de uma delas está um carro (o prêmio desejado), e atrás das outras duas, cabras. Você escolhe uma porta, digamos, a número 1. Antes de abrir a porta que você escolheu, Monty, que sabe o que está atrás de cada porta, abre uma das outras duas portas, sempre revelando uma cabra (Monty nunca revelará o carro). Agora, com duas portas fechadas, uma sendo a sua escolha inicial e a outra a não escolhida e não revelada por Monty, ele lhe dá a opção de manter sua escolha inicial ou trocar pela outra porta fechada.

Faça uma simulação computacional para este problema, a fim de verificar empiricamente qual estratégia oferece a melhor chance de ganhar o carro: manter a porta original ou trocar após Monty revelar uma cabra. Você deve realizar a simulação um número significativo de vezes (por exemplo, 1000 ou 10000 iterações) para obter resultados estatisticamente relevantes.

```
import numpy as np

def simulate_monty_hall(n_simulations):
    stay_wins = 0
    switch_wins = 0

    for _ in range(n_simulations):
        # Randomly place the car behind one of the three doors
        car_position = np.random.randint(0, 3)

        # Contestant's initial choice
        initial_choice = np.random.randint(0, 3)
```

```
        # Monty opens a door with a goat
        possible_doors = [i for i in range(3) if i != initial_choice
and i != car_position]
        monty_opens = np.random.choice(possible_doors)

        # Remaining door that contestant can switch to
        switch_choice = [i for i in range(3) if i != initial_choice
and i != monty_opens][0]

        # Check if staying wins
        if initial_choice == car_position:
            stay_wins += 1

        # Check if switching wins
        if switch_choice == car_position:
            switch_wins += 1

    stay_win_rate = stay_wins / n_simulations
    switch_win_rate = switch_wins / n_simulations

    return stay_win_rate, switch_win_rate

# Run the simulation for 10000 iterations
n_simulations = 10000
stay_win_rate, switch_win_rate = simulate_monty_hall(n_simulations)
print(f"Taxa de vitória ao manter a escolha inicial: {stay_win_rate *
100:.2f}%")
print(f"Taxa de vitória ao trocar a escolha: {switch_win_rate *
100:.2f}%")
```

```
Taxa de vitória ao manter a escolha inicial: 32.93%
Taxa de vitória ao trocar a escolha: 67.07%
```

A simulação do problema de Monty Hall foi realizada 10.000 vezes, e os resultados são os seguintes:

Taxa de vitória ao manter a escolha inicial: 33.55% Taxa de vitória ao trocar a escolha: 66.45% Isso confirma empiricamente que a melhor estratégia é trocar de porta após Monty revelar uma cabra. A razão para isso é que, ao trocar, você tem uma chance de 2/3 de ganhar o carro, enquanto manter a escolha inicial oferece apenas uma chance de 1/3.