Ejercicio 2

Estime el precio de una opción call europea utilizando la fórmula de Black-Scholes a través de la simulación Monte Carlo. Para ello considere lo siguiente:

1. Implemente la fórmula de Black-Scholes para calcular el precio teórico de la opción.
2. Simule las trayectorias del precio de las acciones mediante el movimiento browniano geométrico.
3. Calcule el pago de cada camino y promedielos para estimar el precio de la opción.

```
#!pip install scipy
#!pip install numpy
```

```
Collecting scipy
  Obtaining dependency information for scipy from
https://files.pythonhosted.org/packages/e9/20/2d0561ab54d857365926c5b5
1.11.2-cp39-cp39-win_amd64.whl.metadata
  Using cached scipy-1.11.2-cp39-cp39-win_amd64.whl.metadata (59 kB)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in
c:\users\andre\onedrive\documentos\github\lab4_modsim\.venv\lib\site-
packages (from scipy) (1.25.2)
Using cached scipy-1.11.2-cp39-cp39-win_amd64.whl (44.1 MB)
Installing collected packages: scipy
Successfully installed scipy-1.11.2
```

```
import numpy as np
from scipy.stats import norm

# Parámetros
S = 100      # Precio actual de la acción
K = 100      # Precio de ejercicio
r = 0.05     # Tasa de interés libre de riesgo
T = 1.0      # Tiempo de vencimiento
sigma = 0.2  # Volatilidad
num_simulations = 100000  # Número de simulaciones
```

```
num_steps = 252  # Número de pasos en la simulación (correspondiente a
        días de negociación en un año)


# Fórmula de Black-Scholes para el precio de una opción de compra
def black_scholes_call(S, K, r, T, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma *
        np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    option_price = S * norm.cdf(d1) - K * np.exp(-r * T) *
        norm.cdf(d2)
    return option_price


# Simulación de trayectorias con Movimiento Browniano Geométrico
def generate_gbm_paths(S, r, sigma, T, num_simulations, num_steps):
    dt = T / num_steps
    paths = np.zeros((num_simulations, num_steps + 1))
    paths[:, 0] = S
    for i in range(num_simulations):
        for j in range(1, num_steps + 1):
            z = np.random.standard_normal()
            paths[i, j] = paths[i, j - 1] * np.exp((r - 0.5 *
        sigma**2) * dt + sigma * np.sqrt(dt) * z)
    return paths


# Calcular el precio estimado de la opción usando Monte Carlo
def monte_carlo_option_price(paths, K, r, T):
    option_payoffs = np.maximum(paths[:, -1] - K, 0)
    discount_factor = np.exp(-r * T)
    option_price = np.mean(option_payoffs) * discount_factor
    return option_price


# Calcular el precio teórico de la opción
option_price_theoretical = black_scholes_call(S, K, r, T, sigma)
print("Precio teórico de la opción de compra:",
        option_price_theoretical)


# Simular trayectorias y calcular el precio de la opción usando Monte
        Carlo
paths = generate_gbm_paths(S, r, sigma, T, num_simulations, num_steps)
option_price_monte_carlo = monte_carlo_option_price(paths, K, r, T)
print("Precio estimado de la opción de compra mediante Monte Carlo:",
        option_price_monte_carlo)
```

```
# Comparar precios teóricos y estimados
print("Diferencia entre precio teórico y estimado:",
        abs(option_price_theoretical - option_price_monte_carlo))
```

```
Precio teórico de la opción de compra: 10.450583572185565
Precio estimado de la opción de compra mediante Monte Carlo:
10.412742072257563
Diferencia entre precio teórico y estimado: 0.037841499928001454
```